Proposing a representation model of computational simulations' execution context for reproducibility purposes

Faïçal Yannick Palingwendé Congo

N° d’ordre D. U:

E D S P I C:

UNIVERSITÉ CLERMONT AUVERGNE

ÉCOLE DOCTORALE DES SCIENCES POUR L’INGÉNIEUR DE CLERMONT-FERRAND

T h è s e

Présentée par :

Faïçal Yannick Palingwendé Congo

Pour obtenir le grade de :

DOCTEUR D'UNIVERSITÉ

Discipline : Informatique

Titre de la thèse :

Proposition d’un modèle pour la représentation de contexte d'exécution de simulations informatiques à des fins de reproductibilité

Soutenue publiquement le 19 Décembre, 2018 devant le jury :

David R.C. Hill	Co-directeur de thèse
Mamadou K. Traoré	Co-directeur de thèse
Alexandre Guitton	Examinateur
Kérivin Hervé	Invité
Andrew Davison	Examinateur
Carelyn Campbell	Rapporteur
Francesca Tavazza	Rapporteur

N° d'ordre D. U:

E D S P I C:

UNIVERSITÉ CLERMONT AUVERGNE

ÉCOLE DOCTORALE DES SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

PhD Thesis

Submitted by:

Faïçal Yannick Palingwendé Congo

To obtain the degree of:

PhD in Computer Science

Title of the thesis:

Proposing a representation model of computational simulations' execution context for reproducibility purposes

---

Publicly defended on December 19th, 2018 before the jury:

David R.C. Hill	Thesis Co-director
Mamadou K. Traoré	Thesis Co-director
Alexandre Guitton	Examiner
Kérivin Hervé	Invited
Andrew Davison	Examiner
Carelyn Campbell	Referee
Francesca Tavazza	Referee

To my Father, my Mother, my Wife and my Son

Abstract

Computational reproducibility is an unavoidable concept in the 21st century. Computer hardware evolutions have driven a growing interest into the concept of reproducibility within the scientific community. Simulation experts press that this concept is strongly correlated to the one of verification, confirmation and validation either may it be for research results credibility or for the establishment of new knowledge. Reproducibility is a very large domain. Within the area of numerical and computational Science, we aim to ensure the verification of research data provenance and integrity. Furthermore, we show interest on the precise identification of operating systems parameters, compilation options and simulation models parameterization with the goal of obtaining reliable and reproducible results on modern computer architectures. To be able to consistently reproduce a software, some basic information must be collected. Among those we can cite the operating system, virtualization environment, the software packages used with their versions, the hardware used (CPU, GPU, many core architectures such as the former Intel Xeon Phi, Memory, …), the level of parallelism and eventually the threads identifiers, the status of pseudo-random number generators, etc. In the context of scientific computing, even obvious, it is currently not possible to consistently gather all this information due to the lack of a common model and standard to define what we call here execution context. A scientific software that runs in a computer or a computing node, either as a cluster node, a grid cluster or a supercomputer possesses a unique state and execution context. Gathering information about the latter must be complete enough that it can be hypothetically used to reconstruct an execution context that will at best be identical to the original. This of course while considering the execution environment and the execution mode of the software. Our effort during this journey can be summarized as seeking an optimal way to both ease genuine access to reproducibility methods to scientists and aim to deliver a method that will provide a strict scientific numerical reproducibility. Moreover, our journey can be laid out around three aspects. The first aspect involves spontaneous efforts in collaborating either to bring awareness or to implement approaches to better reproducibility of research projects. The second aspect focuses in delivering a unifying execution context model and a mechanism to federate existing reproducibility tools behind a web platform for World Wide access. Furthermore, we investigate

applying the outcome of the second aspect to research projects. Finally, the third aspect focuses in completing the previous one with an approach that guarantees an exact numerical reproducibility of research results.

Keywords: Reproducibility, Platform, Web, Tools, Caching, Artifact, Interoperability, Integration, Execution, Environment.

Résumé

La reproductibilité en informatique est un concept incontournable au 21ème siècle. Les évolutions matérielles des calculateurs font que le concept de reproductibilité connaît un intérêt croissant au sein de la communauté scientifique. Pour les experts en simulation, ce concept est indissociable de celui de vérification, de confirmation et de validation, que ce soit pour la crédibilité des résultats de recherches ou pour l'établissement de nouvelles connaissances. La reproductibilité est un domaine très vaste. Dans le secteur computationnel et numérique, nous nous attacherons, d'une part, à la vérification de la provenance et de la consistance des données de recherches. D'autre part, nous nous intéressons à la détermination précise des paramètres des systèmes d'exploitation, des options de compilation et de paramétrage des modèles de simulation permettant l'obtention de résultats fiables et reproductibles sur des architectures modernes de calcul. Pour qu'un programme puisse être reproduit de manière consistante il faut un certain nombre d'information de base. On peut citer entre autres le système d'exploitation, l'environnement de virtualisation, les diverses librairies utilisées ainsi que leurs versions, les ressources matérielles utilisées (CPU, GPU, accélérateurs de calcul multi cœurs tel que le précédent Intel Xeon Phi, Mémoires, ...), le niveau de parallélisme et éventuellement les identifiants des threads, le statut du ou des générateurs pseudo-aléatoires et le matériel auxquels ils accèdent, etc. Dans un contexte de calcul scientifique, même évident, il n'est actuellement pas possible d'avoir de manière cohérente toutes ces informations du fait de l'absence d'un modèle standard commun permettant de définir ce que nous appellerons ici contexte d'exécution. Un programme de simulation s'exécutant sur un ordinateur ou sur un nœud de calcul, que ce soit un nœud de ferme de calcul (cluster), un nœud de grille de calcul ou de supercalculateur, possède un état et un contexte d'exécution qui lui sont propres. Le contexte d'exécution doit être suffisamment complet pour qu'à partir de celui-ci, hypothétiquement, l'exécution d'un programme puisse être faite de telle sorte que l'on puisse converger au mieux vers un contexte d'exécution identique à l'original dans une certaine mesure. Cela, en prenant en compte l'architecture de l'environnement d'exécution ainsi que le mode d'exécution du programme. Nous nous efforçons, dans ce travail, de faciliter l'accès aux méthodes de reproductibilité et de fournir une méthode qui permettra d'atteindre une reproductibilité

numérique au sens strict. En effet, de manière plus précise, notre aventure s'articule autour de trois aspects majeurs. Le premier aspect englobe les efforts de collaboration, qui favorisent l'éveil des consciences vis à vis du problème de la reproductibilité, et qui aident à implémenter des méthodes pour améliorer la reproductibilité dans les projets de recherche. Le deuxième aspect se focalise sur la recherche d'un modèle unifiant de contexte d'exécution et un mécanisme de fédération d'outils supportant la reproductibilité derrière une plateforme web pour une accessibilité mondiale. Aussi, nous veillons à l'application de ce deuxième aspect sur des projets de recherche. Finalement, le troisième aspect se focalise sur une approche qui garantit une reproductibilité numérique exacte des résultats de recherche.

Mots clés : Reproductibilité, Plateforme, Web, Outils, Enregistrement, Artéfact, Interopérabilité, Intégration, Exécution, Environnement.

Acknowledgments

This journey has inevitably contributed to building who I am today. However, it is not because of Science only if I may say. I have been lucky to who supervised it. Even luckier to where I was allowed to carry it. And I am the luckiest to how much support I was awarded.

Prof. David R. C. Hill has been more than a teacher to me since I laid foot in the former French engineering school Institut Supérieur d'Informatique, de Modélisation et de leurs Applications (ISIMA) seven years ago. I was lucky to have learned from him during my engineering curriculum and even more to have him as my lead PhD co-supervisor. His always up to date large knowledge of various spectrum of domains in Science and beyond has always fascinated me and always will. I still don't know how he does it but can only hope I will grow scientifically to his standard.

Dr. Mamadou K. Traoré and I have met each other only recently to my regret. I wish I had known him longer. But no matter, his help as my co-supervisor has been more than essential in my most critical moments. I will always be in debt to what he did for me. I can only hope that after his duty to me as a supervisor and the end of this journey, I will still have a tiny bit of his time in the new journey I will rush into. His dynamism and perspicacity are truly exceptional. I do not think there are many people like him among us.

Everyone I have ever met that went through this journey before me never skipped telling me about the hard times and how vital it was to have the necessary support to keep focusing on the goal. However, they never told me how they got through it themselves. Feeling the private and sensitive nature of these moments, I never dared ask but hoped I will be awarded the same treat. Here, I will share mine because I think it is worth doing so but in very short words, so I do not spoil others. I would assume everyone have their own experience.

First, family is all indeed and even more in the hard times. My father, mother and my wife from my perspective, were my rocks. Consciously and sometimes unconsciously they all went to great length to be there for me in those times. Beyond our bond before this journey, I celebrate them here in these short lines. The luck of my life is to be the son of first two and the husband of the last respectively. Moreover, I celebrate my son Naël for the timing of his arrival. All the nights I have spent finalizing this manuscript would have been lonelier without his company.

Second, there are people without whom, this journey would have not even begun to start. My employers and colleagues at the National Institute of Standards and Technology (NIST) in the United States of America have made my journey one of the most exiting anyone could have. I am truly grateful to the support, trust and opportunity given to me and can only hope what comes out and what is to come is worth their efforts. I would like to cite Dr. Carelyn Campbell, Dr. James Warren, Dr. Daniel Wheeler and Dr. Jonathan Guyer and all my friends over CTCMS for everything they did for me.

Last and not the least, this journey would have not had the success it had without the staff at Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS), École Doctorale des Sciences Pour l'Ingénieur (EDSPI) and the exceptional leadership of their respective directors. The guidance and support I received was beyond my expectations. I am in dept to these institutions and the people that make their reputations.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AFRL	Air Force Research Laboratory
ANN	Artificial Neural Network
API	Application Programming Interface
AWS	Amazon Web Services
CHiMaD	Center for Hierarchical Materials Design
CPU	Central Processing Unit
DevOps	Development and Operations
DFT	Density Functional Theory
EDSPI	Ecole Doctoral des Sciences Pour l'Ingénieur
FACT	Facility for Adsorbent Characterization and Testing
FF	Force-fields
GPU	Graphical Processing Unit
HTTP	HyperText Transfer Protocol
ICE	Integrated Computational Environment
JARVIS	Joint Automated Repository for Various Integrated Simulations
JSON	JavaScript Objection Notation
LLNL	Lawrence Livermore National Laboratory
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
MATIN	MATerials Innovation Network
MNIST	Modified National Institute of Standards and Technology
MRS	Materials Research Society
NIST	National Institute of Standards and Technology

OISM	Office of Information Security and Management
OS	Operating System
OSF	Open Science Framework
QI	Quantification de l'Incertain
RESTful	Representational state transfer
SFTP	Secure File Transfer Protocol
SHA-256	Secure Hash Algorithm 2
SQLite	Structure Query Language lite
SRM	Standard Reference Model
TTC	Transformation Tool Contest
TU/e	Technische Universiteit Eindhoven
UCA	Université Clermont Auvergne
UQ	Uncertainty Quantification
URL	Uniform Resource Locator
VASP	Vienna Ab initio Simulation Package
XML	eXtensible Markup Language
YAML	Yet Another Markup Language

Chapter 1 – GENERAL INTRODUCTION

1.1 ON THE MATTER OF REPRODUCIBILITY

One of the most currently trending crisis in Science is labelled under issues with the generic concept of reproducibility [**Baker 2016**]. As a key element of the scientific method itself, impeding on the latter shakes Science at the core of its success. One of the main causes is probably the growth in complexity and the diversity of systems and methods. Moreover, fraud in Science has also been identified as a cause simply because of the relentless hunt for more fame and all its resulting benefits [**Eisner 2018**].

Anyhow, the fact is that Science has evolved and still is. It is not anymore at its early ages of pure intellectual endeavors. Today, numerous of its results are systematically used in life critical missions such as curing disease, sending mankind out of earth, piloting autonomous cars or protecting financial, private and sensitive data [**Andel et al. 2012**]. The cost of a scientific fraud or any failure to corroborate back in the golden age of Science (between 8th and 14th century) had more chances to only cause discredit to the inculpated scientist. Today, the consequences of fraud and scientific mistakes can have far greater damages. In fact, a fundamental trust in Science is questioned by the general public which is mostly affected by using the applicative end solutions of scientific results [**Smith 1997**]. Now, we are in the ages where the scientific method, the core success of Science, must revise the safeguards of its reproducibility requirement. For Science that cannot be recreated consistently is not Science. Moreover, Science that cannot be corroborated effectively is not Science.

Science today has diversified itself. Theoretical and experimental are its two eldest forms [**Stodden 2017**]. They have been motivating each other since the dawn of Science. By the early 19^{th} century advancement in these two forms has paved the way for a third form of scientific inquiry. Motivated by the need to automate calculus and equation resolutions always faster and faster, a newer form of Science up to the task has been introduced: computational. This new form has proven itself to bridge the two previous. Also, despite its unicity, it can be seen as a productivity booster for these two and it has impacted and boosted the evolution of more 'traditional' Sciences [Kurzweil 2004].

However, independently of the form in which a result is being presented, Science requires that it must be indefinitely provable [**Burke 1962**] in the same conditions. The power of theoretical Science is in its hundred percent intellectual nature. With the appropriate theoretical background (ground truths), some paper, means of writing and enough time, any well-aware scientist can corroborate any result. A result using this form is proved with theorems, axioms, data and results by any logical means possible in the constrained construct. Despite its growth in complexity and sophistication, theoretical Science has preserved this level of abstract elegance in its corroboration mechanism. Unfortunately, this is not the same neither for experimental nor is it for computational. With these two forms, the complexity incurred by the furthering of Science is correlated to the growing challenges of their diverse corroboration mechanisms. In computational Science for example, due to enormous differences in computers architectures and operating systems, being able to run a simulation in one computer does not always translate well to any other computer or operating system. Furthermore, within the same system, being able to run a simulation now does not imply that it will still be the case at different points in time. Numerous things occur in computers today and some of these things are likely to disrupt the possibility to successfully run a simulation again. To make this more challenging, with computer technology changes occur quickly. Five years are enough to see hardware architectures become discontinued, file formats unsupported and so on.

Scientists have taken the matter to their own hands. Around the world, teams of scientists are developing software to support the process of making computational Science results corroboration easier. These tools are created to reduce the effects of the growing complexity within computational environments. Their main approaches are: detailed documentation, automation and guided corroboration. Thus, instead of being faced with thousands of ways to be lost in various modes of failing to corroborate a result, these tools and methods either fully or partially do it on behalf of the scientist or guide the scientist through the least amount of effort to realize the corroboration.

The present document communicates the outcome of a deep overview of modern day corroboration of scientific results. Moreover, our work focused in how to best address some of the persistent problems of the latter. Thus, we address the meanings of the notions used by

scientists today through the core reason of this entire investigation. Extensive knowledge on the current state of the art in the matter of corroboration is compiled and reviewed at the discretion of the reader. Also, we describe the two main results of this research. We believe they are significant contributions to the body of Science in the matter of scientific results corroboration.

Mastering all different aspects of corroboration has the privilege of putting one at the stage of the unknown and spikes the continuous fuel of curiosity to try and drive advancement in an even more significant manner. We assume this to be the "fate" of the cutting-edge scientist. However, there is another side to the one of being at the leading line of a scientific domain. The frustration of the limitations in every solution and approach we know so far is most pressing. In fact, we quickly grasp the fact that we clearly leave in a world of compromises. Indeed, most of these existing solutions will be suitable in very specific situations and be totally inadequate in others. We welcome the reader into an activity that will elude the specifics of these words may they appear too abstract at the moment. To start, we invite the reader to the next section for a slightly more detailed presentation of the content and the structure of the document.

1.2 THESIS OUTLINE

In the next chapter, we elaborate on the problematic that gives sense to this adventure. More specifically, we first take on the details of the current nuances to the notion of reproducibility and it ties to the general notion of scientific outcome corroboration. Then, we develop the fundamental problems that seriously imped the awareness of reproducibility issues and the access to the current solutions expressed today in the form of tools and web platforms. Additionally, we deem appropriate in this chapter of problematic to share the motivations behind the problematic statement of this thesis. Thus, we propose to elaborate on some of the most persistent problems in reproducible research and the reasons that motivated the early motion of this thesis. Furthermore, we invite the reader into a deep overview of the current state of the art in term of reproducibility and in general, the corroboration of scientific results.

In chapter 3, we present contributions in the literature aimed towards bettering the reproducibility of research results. It defends its bipartite view of these contributions. In fact, the first section overviews contributions in terms of software designed for general-purpose

reproducibility issues. Such issues scope roughly, attempting to successfully run a previous computation anew without a runtime crash. Thus, a result should be obtained. Moreover, the result is expected to be within acceptable error scales. Also, this first section explores the current methods used to perform a numerical assessment with the goal to reach results that are invariable to changes. The second section presents the landscape of reproducible research platforms that go a step further than the previous ones. In fact, a small fraction of these platforms provides an outline of solutions to the problematic statement presented in the previous chapter. However, as we show in this chapter, none of the existing solutions approaches the problems of interest in this document at the same degree of completeness as the results of this research. This brings us to the main reason behind the two major contributions during this thesis. They are respectively developed in the following chapters.

In chapter 4, we introduce the first result of this research in response to three statements of this research problematic. Moreover, this result is engineered in accordance to our first motivation. In this chapter, we present a web platform named CoRR (Cloud of Reproducible Records). This platform presents itself as a sound solution to the three first problems stated in this document by being a feature augmenting gateway for tools and platforms alike. Those features include: web capability, scientific collaboration, inherent dissemination and a promise of interoperability. Thus, it surpasses any of the existing solutions in general-purpose reproducibility and the current platforms landscape by providing an all in one answer to most of the problems of interest in this thesis. However, it does not address the last of the four problematic statements described in this document in chapter 2.

Consequently, in chapter 5, we detail a new method that can guarantee an exact reproducibility of research results. Therefore, this result implemented as a library named Num-Cache is a viable solution to the last unsolved problem by our first result, presented in the previous chapter. In fact, most reproducible research methods are currently either using numerical approximations to reduce variations or in general just attempting to achieve execution without a runtime crash. This infers that results from ulterior runs will most likely drift from the original due to many untracked changes such as scheduling and hardware tricks. Num-Cache comes in this chapter as a complementary effort to the one of the previous chapter. By combining these two we are able

to address some of the most persisting problems in reproducible research. Thus, we recommend scientists in the next chapters that obtaining a complete solution to the problems eluded in this document involves using Num-Cache in their simulations tracked by a CoRR supported software and obviously connected to a running instance of the latter. As a result, their computations will have better chances of being reproduced, shared and improved through collaborations with other scientists. These two contributions have been applied during their development and after their current experimental releases. The following chapter elaborates on these applications.

In chapter 6, we briefly narrate a few of the collaborative activities that occurred with the two results of this thesis. First, we present the integration activities involving the first three supported tools in the current experimental version of CoRR. These tools have been carefully chosen due to their audience, their presence and the impact of CoRR in pushing them further with its features. Second, we list five of the most critical use cases in collaborative contributions involving mostly CoRR or the solicitation of the knowledge absorbed during this thesis. The main applications to Num-Cache have been presented in the chapter 5 and at SummerSim 2018 conference in Bordeaux, France. However, complete and directly applicable, the two results presented respectively in the previous chapters have limitations of their own. We open discussions into the latter and beyond in the next chapter.

Chapter 7 discusses the strength and the current limitations of CoRR and Num-Cache. Moreover, we debate on the other problems not being handled in this thesis against which our solutions are not tailored for. Such cases are the notion of reproducibility not meaning correctness, as the latter requires more. Thus, we open discussions into the crucial need for independent verifications which indeed are the ultimate solution to approaching the unanimous correctness of research results. Yet, the unavoidable problems of uncontrolled runtime variations and reproducibility at Exa-Scale are also addressed in these discussions as they are both challenging our current solutions.

Chapter 2 – PROBLEMATIC STATEMENTS

2.1 ON THE CURRENT MEANINGS OF CORROBORATION

The credibility of research results has never been as much scrutinized as now [**Franzen 2016**]. It almost seems as there was a grace period of negligence that gave room to the inadmissible issues we are facing today [**Bhardwaj 2015**]. Issues that every reproducible research advocate currently points at. Thus, we are witnessing growing efforts from institutions, journals and individuals to unite for reproducibility enforcement [**McNutt 2014**]. Therefore, it is vital for the reader to grasp the great deal of importance behind the concept of reproducibility. In other words, how was reproducibility framed and why is it so critical to the success of Science (Episteme), shown in Figure 2.1?

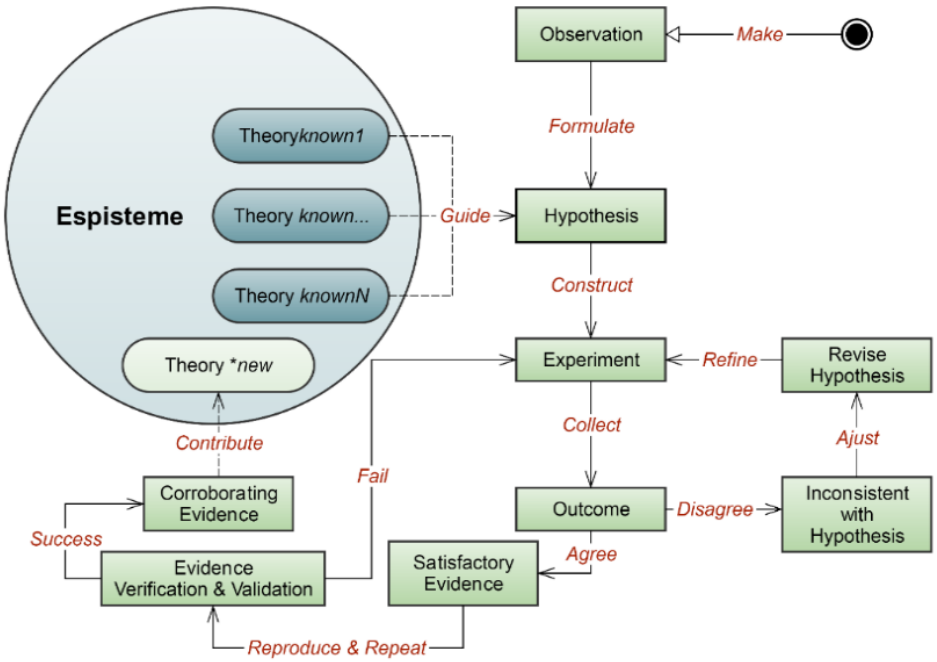

Figure 2.1 The scientific method: The scientific inquiry is expected to leverage previous contributions to Science and follows a standard path of investigation to reach satisfactory evidence that could lead to the addition of new knowledge.

Our journey to the origin and foundation of reproducibility takes us back to two of the main elements of success of Science. Science is thought to be the most optimal form of reasoning according to Stanford Encyclopedia of Philosophy [Andersen and Hepburn 2016]. One of its main

elements of success goes deep into its standard guideline for scientific inquiry depicted in Figure 2.1: The scientific method.

Since Aristotle's Organon [Aristotle et al. 1938], many forms of the scientific method have been used by theoreticians, experimentalists and nowadays computationalists [Andersen and Hepburn 2016]. With it, scientists can tool themselves to achieve new discoveries knowing that the outcome will be mostly welcomed. Yet, why should we trust the scientific method? This question takes us on to the second main elements of Science's success that we want to introduce here. The core feature of human reasoning is the intrinsic need for new knowledge to rely on older ones. This is how Science has incrementally grown to converge toward more sophistication or by replacing false understandings by more robust findings. The continuous accumulation of discoveries in Science have laid the path towards ever more fundamental knowledge [**Bird 2008**]. Pr. Albert Einstein formulated his theory of special relativity from the difficulties faced by Pr. James Clerk Maxwell in his tremendous body of contribution to our current knowledge of Electromagnetism [**Snyder 2000**]. Yet, what makes Pr. Maxwell's, and all known scientific discoveries universally accepted today is the most critical requirements of the scientific method: corroboration. Anyone at any time with the same tools and workflow of inquiry following the scientific method must be able to consistently corroborate a previously accepted outcome. Corroboration is strengthened when the use of different tools or approaches leads to the same scientific conclusions. The corroborative feature of the scientific method is the foundation for trust through which new scientific knowledge is being chained to old ones and majorly accepted as part of a body of contribution to Science [**Kurt and DeSalle 2009**].

The latter feature of the scientific method has evolved to undertake many forms today. Science has advanced and so did the tools for performing inquiries. We are not anymore at the times when there was no or very limited number of tools to achieve the same outcome. Today, we thankfully have more tools than ever before, from more manufacturers, with more interface types and based on more designs. As such, corroborating discoveries has increased in complexity.

In the experimental Sciences, laboratory machines for achieving equivalent tasks have increased in number. However, standards haven't got all the way across manufacturers processes. In fact, fierce competition has brought tremendous differences in how machines are used, how they

function and how their outputs are structured. Hence, the need for differentiation and secrecy have created an inconsistent ecosystem of disconnected marketplaces of brands. Consequently, using any equivalent tools in the scientific method with the hope to obtain a previous outcome is not guaranteed. Moreover, the time incurred in achieving equivalent manipulations on machines with different interfaces and design philosophies is considerable. An appropriate example is probably the major difficulty faced by Dr. Laura Espinal in the NIST Facility for Adsorbent Characterization and Testing (**FACT** (**https://www.nist.gov/mml/fact** accessed September 28, 2018)) lab. It is a facility commissioned to provide state-of-the-art measurement capabilities for impartial, accurate testing and characterization. Measurement equipment includes volumetric instrumentation, gravimetric instrumentation, and combined systems for multi-component gas mixtures. Absorbents include carbon dioxide, methane, hydrogen, nitrogen, helium, water vapor, and toluene. In adsorptive material characterization, it had been challenging to assure consistent, reliable measurements between laboratories. There are often discrepancies in isotherm data from "round robin" studies mostly pointing to the use of instruments from segregated manufacturers and designed around different measurements approaches.

In the computational Sciences, problems persist despite the non-zero-sum games enforced by major stakeholders [**Kerber and Schweitzer 2017**] for the sake of interoperability and portability. In fact, the diversity of computer hardware, operating systems, formats, protocols, specifications and software designs is widening the complexity involved in corroborating a research simulation. Each of these aspects of computers evolves separately in its own confine challenges, requirements and obsolescence paradigms. While hardware manufacturers battling with the laws of physics push for smaller and more powerful machines, software architects are unleashing their creativity in building more sophisticated applications that will leverage the power given by newer machines. Furthermore, the amount of dependencies in today's software and their changes per release cycle simply surpasses our capability to effectively catalog them manually. Figure 2.2 shows the number of dependencies added or removed in 89 python packages (colored) as their versions increase.

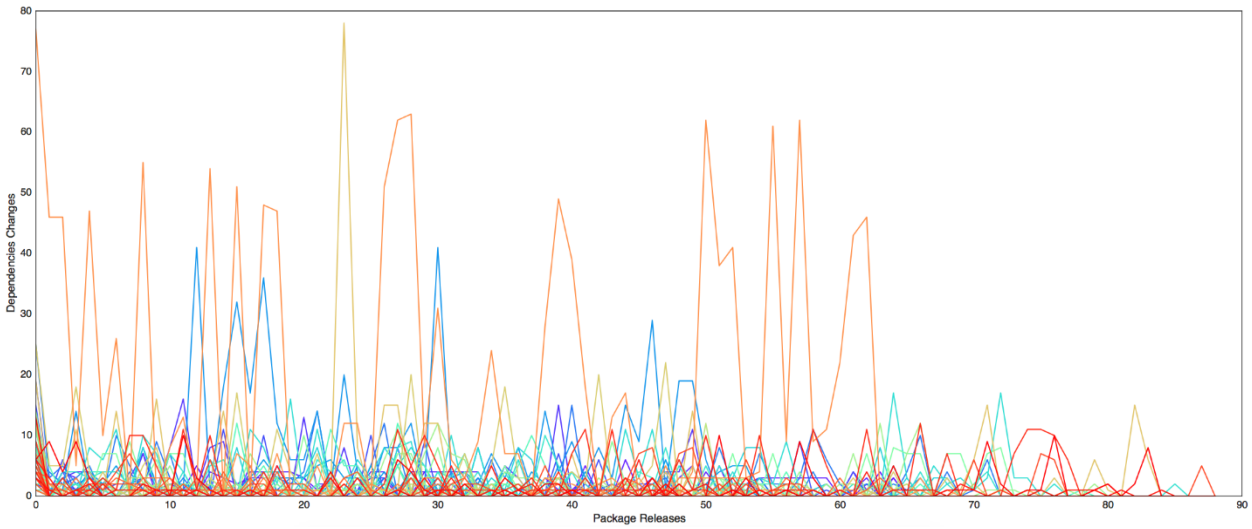

Figure 2.2 Dependencies removals and additions in major python packages on PyPi: The change in dependencies (y axis) per package release (x axis) allows us to appreciate its globally chaotic nature.

The previous figure is based on data from packages listed on PyPi (**https://pypi.org** accessed September 28, 2018) with at least 50 release cycles. This data clearly shows a glimpse into how the dependency growth and variations in modern software can be very challenging to monitor without specialized automated tools.

The overall complexity incurred with the recent technological advances requires more nuances to express the early meaning of the term 'corroboration', which alone has become too ambiguous. Such nuances express how strongly the corroborator must comply with the ever-diversifying tools and core parts of the scientific method that lead to a specific outcome. We have enumerated three major nuances:

Definition 1.1. The strongest meaning of corroboration requires the usage of the exact same tools involved in the scientific method with the same scientific approaches that lead to a previous outcome.

Definition 1.2. The weakest meaning of corroboration focuses in obtaining a previous outcome at all cost. Thus, the tools or more often the scientific approaches are expected to differ from a previous inquiry that unveiled the outcome.

Definition 1.3. The closest meaning of corroboration of yore scopes any attempt to reach to a previous outcome by following the same scientific approaches yet tolerating variations in the tools used across the scientific method.

Terms such as **reproducibility**, **repeatability**, **replicability**, etc. have been controversially recycled across different scientific communities to correspond to one of the three nuanced definitions listed above. According to Webster's (**https://www.merriam-webster.com** accessed September 28, 2018) Third New International Dictionary, these three terms have the following meaning:

Definition 1.4. Reproducibility is the capability to cause to exist again or anew; or to cause to be or seem to be repeated.

Definition 1.5. Repeatability is the capability to say or state again; or to say over from memory.

Definition 1.6. Replicability is the capability to copy, duplicate; or to produce a facsimile of an original work.

The light differential ambiguity between these terms and the ongoing terminology adjustments between communities, groups and individuals have tasked us early on in this thesis. In fact, how can the debate be effectively contributive if we mean different things with the same terms? We will be discussing this aspect thoroughly in chapter 3.

While the nuances in modern day corroboration are still not being dramatically experienced unanimously, the current issues in corroborating scientific outcome in general is rising attention and gaining momentum [**Munafò et al. 2017**]. Hence, more institutions are funding scientific teams to develop methods and tools that ensure the corroboration of their results [**LeVeque et al. 2012**].

A typical modern research outcome does not contain the pedigree of tools used in the scientific method workflow that produced it. Yet, having this information is the only sustainable way to attempt any corroboration. Thus, a recommended good habit schooled to new scientists is to always record this pedigree in their lab notebooks. While this might have been appropriate in the past, it is currently becoming humanly impossible for a scientist to manually account for everything that contributed to produce a specific result. Hence, scientists either find themselves

not being able to corroborate their own results or others at another point in time. Moreover, some scientists find themselves not being able to corroborate results that others are able to corroborate. It is becoming obvious that manually recording the pedigree of a research inquiry even with the guidance of the scientific method is a closing dead end. The utterly consequence is that most scientists end up fighting their way back to reconstruct the pedigree of tools and steps that lead to one of their own results or someone else's.

The currently developed tools to ensure corroboration of scientific results focus mainly on how to best capture and store the pedigree of tools and methods used in the scientific method. While some tools look at the entire pedigree, others drive their interests only towards key steps in the pedigree. Independently from the specifics of the techniques used, one common and unchanged characteristic is automation. Most of these tools provide a guided or automatic way of capturing the pedigree. Yet, since these tools have been developed without a consensus, they each require training to be properly used. In fact, what is important to catalog in the pedigree for one team in an institution is not the same for one in another institution or even in the same one for that matter.

This thesis aims in, first, bringing awareness as early as possible towards the need of tools to shield scientists from the current hidden complexities of trying to manually corroborate a result. For the sake of awareness, we dare to compare the importance of reproducibility to version control [**Koc and Tansel 2011**]. In fact, scientists not aware of the ecosystem of version control tools that exists today will do as everyone of us did. They will spend a tremendous amount of time compressing and datetime naming the same folders over and over. Keeping up with this scheme quickly drives anyone weary, especially when faced with trying to manually perform a collaborative merge of different modifications. This is barely the case now, everyone knows and talk about version control. Sadly, reproducibility is at this early situation faced by version control.

Secondly, we have taken to heart to address a direct consequence of the thriving growth in terms of tools and services in support of reproducible research. In fact, such a growth has given birth to a disconnected web of solutions that do not talk to each other nor interoperate with each other. Consequently, we worry that new scientists getting into reproducible research will be overwhelmed. Furthermore, scientists already well versed in this aspect of Science, will fear being

trapped in one tool. In fact, since there is no guarantee which tool will not lose support and survive, it is a bit of a gamble to adopt one and face the fact that all the records of one's scientific experiments will be obsolete and require significant work to be ported to another.

The body of knowledge unearthed during this research is barely scratching at the surface of what is a major concern now. Thus, we dare hope that the spark of this present endeavor will ignite a flame that will be carried on in a fashion that soon enough, searching for the keyword "reproducibility" on any major search engines such as Google, Bing, etc. will recommend tools instead of the current references to publications about its challenges and terminology ambiguity as compared to version control.

We invite the reader to perform a quick sentiment overview from the results of searching for both keywords blocks "version control" and "reproducibility" in the Google search engine. It is interesting to see how terms used for reproducibility tend to indicate more dynamism, more movement than those for version control. It is as if, from a simple search, one could have a sneak peek into the entropy of scientific debates and concerns regarding specific subjects [**Demartini and Siersdorfer 2010**] using technologies such as **IBM Watson** (**https://www.ibm.com/watson/** accessed September 28, 2018). We gladly hope to be wrong in the upcoming years.

STATEMENTS

In the previous section, we have brushed up the origins of new meanings behind the term reproducibility: corroboration. Then, we have pointed to the terminology mismatches and disagreements among scientists. Additionally, we have made the parallel between the phenomena of reproducibility global awareness struggle to the early times of version control. Despite the funding and involvement of more institutions, we have come to the fact that the current solutions are mostly missing fundamental features of the modern-day scientist environment. These features are listed as statements as following:

P_1. ***Lack of adequate means of reaching out to Scientists:***

It's one thing to know that what one is experiencing is a reproducibility problem instead of a lack of proper usage of tools. It is another thing to gather awareness that the problem is fundamental

and that there are solutions in the form of software and services to help. We are living today in a connected world through Internet. Most of us get informed mainly by reading, listening or watching contents from web multimedia. The success of digital advertising through Internet is a crucial metric to assess how important web presence is. Awareness on the existence of reproducibility support tools is almost absent simply because very few of the developed tools are designed with the proper dedication to this philosophy. Consequently, most informed reproducible research advocates feel the frustration that in general their audiences may grasp the problems they are addressing but have no clue to what **Sumatra** [**Davison 2014**], **ReproZip** [**Chirigati et al. 2016**], **CDE** [**Guo 2012**], etc. are. There has been tremendous ingenuity poured into these projects. Surely their reputation can be boosted.

P_2. ***Absence of a standard or an interoperability feature between the current tools:***

The rich variety of tools developed proves that the issues of reproducibility are taken seriously. Yet, the growing number of tools is bringing an overwhelming breath of confined designs and internal representations. In fact, as of today none of the tools have a process to turn what its representation of a reproducible capture is to the one of another tool. It is on the other end crucial to note that we are living in a world in which obsolescence is a de facto parameter of our technological advances. Hardware architectures, software designs, file formats and much more have been judged obsolete in the past and thus not supported anymore on today's systems. As such, standards for following common principles for the sake of sustainability in research work is unpriceable. However, when standards have not reach proper maturity, today's software must focus in being interoperable with others as much as possible and as early as possible. The fear of being trapped with obsolete data unusable with other software is effectively present. Today's software users will naturally go to the tools that provides the most useful features and accommodates the best with other software. This problem goes back to the previous one on the aspect of users' adoption. In fact, what is the benefit of using a specialized tool with a doubtful future if one can produce an ad hoc scheme proper to a situation of interest that one understands and is able to shift at will?

P_3. *Scarcity of collaborative features in a collaboration driven world:*

The issues of reproducibility as presented in the previous section under the more generic term of corroboration is inherently a collaboration drama. In fact, corroborating a result mostly involves others considering how they could appreciate, vet and use it. Additionally, the trust in Science comes from its collaborative nature. New discoveries are built on top of the successes and failures of old ones through communication. As such, tools supporting reproducibility must be built with collaborative features. This is not currently the case in most core tools. And for those who are providing such features, they suffer from the previous problem of interoperability which makes the collaboration very confined to an isolated ecosystem. It's a fundamental thing to be able to collect reproducible artifacts of research inquiries for personal use. Yet, it is another critical one to be able to appropriately share and collaborate around such artifacts in our collaboration driven world. Our technological advances tend to strongly support that we are moving toward a de facto of the latter.

P_4. *No solution to exact reproducibility due to Numerical precision issues:*

Numerical precision is the key to advance computation. Without it, we are trapped at a level of accuracy that does not allow us to unearth the secrets of the infinitely small, infinitely fast, infinitely big and infinitely slow. And we have much more to learn at these scales. Yet, we must keep in mind that computers approximate our theoretical calculations which suppose a perfect result. In fact, the reality is that our current CPU architectures are showing the signs of their limitations. The issues of numerical precision due to the loss of the associativity property when computing big and small number is an appropriate example. Equations 1.1 to 1.11 describe a computational problem in which the sum of a permuted bag of numbers does not always return the same results.

Let *E* be the union of bags of numbers. $\left\{\frac{1}{1000}\right\}^{10}$ is a bag of 10 occurences of the number $\frac{1}{1000}$.

Meaning, $\left\{\frac{1}{1000}\right\}^{10} = \left\{\frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}, \frac{1}{1000}\right\}$.

$$E = \left\{\frac{1}{1000}\right\}^{10} \cup \left\{\frac{1}{100}\right\}^{9} \cup \left\{\frac{1}{10}\right\}^{9} \cup \{1\}^9 \cup \{10\}^9 \cup \{100\}^9 \cup \{1000\}^9 \quad (1.1)$$

We want to generate 10000 ordered bags from *E* by permutating its elements. Thus, E_i is an ordered bag and the permutation number i of *E*. Here, A ∪ B represents the union of the two bags A and B.

$$\forall i \in [0,9999];\ E_i = Permutation(E,i) = \{\sigma^i(x) | x \in E\}\ (1.2)$$

Then, we compute S_i the sum of all numbers of each of the generated ordered bags E_i.

$$S = \{\forall i \in [0,9999]; S_i = \sum_{0}^{|E|-1} \{E_i\}\ (1.3)$$

Mathematically, the permutation should not impact the result. We have indeed constructed *E* purposely so that its mathematical true sum be exactly 10000. The following equations explain how this value is obtained. S_{true} represents the expected mathematical result of S.

$$S_{true} = \sum_{0}^{|E|-1} \{E\}\ (1.4)$$

$$\sum_{0}^{|E|-1} E = \sum_{0}^{9} \frac{1}{1000} + \sum_{0}^{8} \frac{1}{100} + \sum_{0}^{8} \frac{1}{10} + \sum_{0}^{8} 1 + \sum_{0}^{8} 10 + \sum_{0}^{8} 100 + \sum_{0}^{8} 1000\ (1.5)$$

$$\sum_{0}^{|E|-1} E = \frac{1}{100} + \frac{9}{100} + \frac{9}{10} + 9 + 90 + 900 + 9000 = 10000\ (1.6)$$

Computationally, we have also crafted *E* to prove the point that the permutation has an impact in the sum of Floating-Point. S_{comp} shows the actual computation evaluation of each elements of the bags in Equation 1.1. The next paragraph gives the result of S_{comp} for the 10000 permutations.

$$S_{comp} = \{1e^4\}^9 \cup \{1^{\Theta 1}e^4\}^{424} \cup \{1^{\Theta 2}e^4\}^{3394} \cup \{1^{\Theta 3}e^4\}^{5203} \cup \{1^{\Theta 4}e^4\}^{970}(1.7)$$

$$1^{\Theta 1} = 1.(0)^{15}1819\ (1.8) \qquad 1^{\Theta 2} = 1.(0)^{15}3638\ (1.9)$$

$$1^{\Theta 3} = 1.(0)^{15}5457\ (1.10) \qquad 1^{\Theta 4} = 1.(0)^{15}7276\ (1.11)$$

S_{comp} shows the computational result done on an Intel ® i7 CPU.

When aggregating the different results across the possible permutations as shown in Figure 2.3, we unveil the Achilles' tendon.

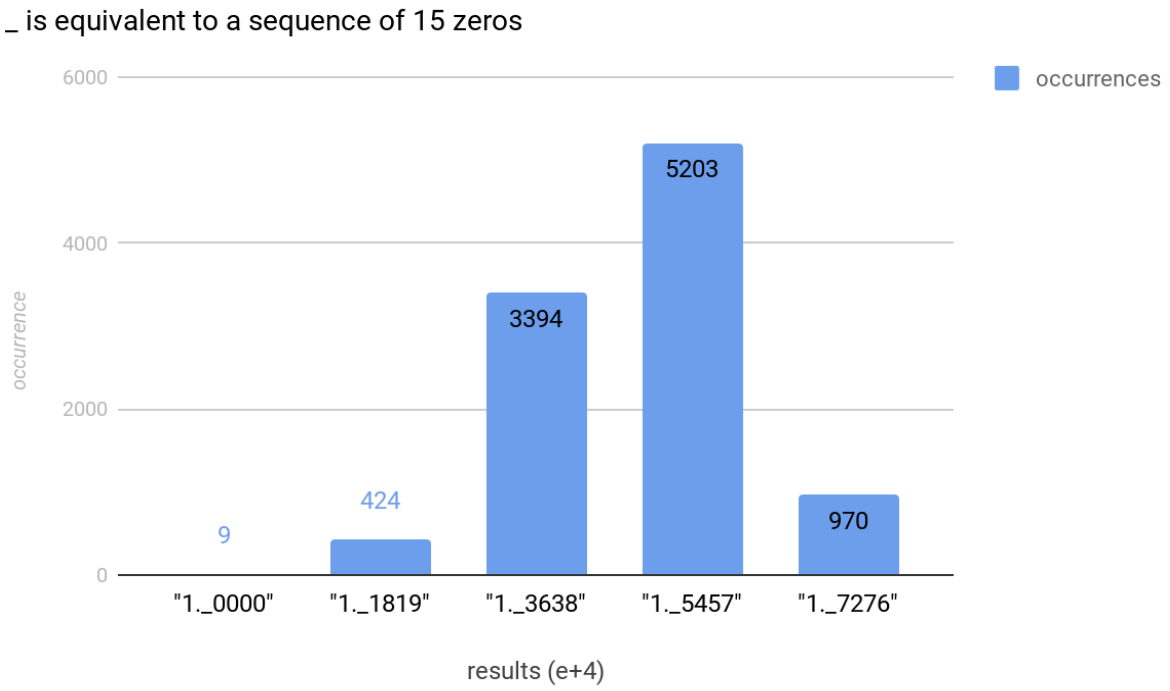

Figure 2.3 Results of the sum in Equation 1.1 for 10000 permutations: The occurrences of the results from the sum in the 10000 permutations clearly shows that in the case of out of order computation [**Zitzlsberger 2014**], this specific sum gives a 0.09% chance to get the correct result.

Only 9 out of the 10000 permutations return the correct mathematically result of 10000. Currently, none of the tools available for supporting reproducibility provide an appropriate solution to these specific issues of numerical precision.

The use of multi-core architectures and networks of many computers have been the way to solve problems involving massive amount of computation through programming models such as MapReduce [**Dean and Ghemawat 2004**]. As shown in the previous example, the order in which numerical operations are carried out is important when dealing with Floating-Point numbers [**Goldberg 1991**]. When not taken care of and thus carried through more complex functions, precision errors will grow and thus make results significantly drift from what should be expected. Furthermore, optimization techniques available on processors such as the Intel ® Xeon Phi's out of order cores that have proven to significantly speed up computations are not recommended

[**Zitzlsberger 2014**] when accuracy is the goal. The order of the numerical operations becomes non-deterministic which leads to different results from a run to another. In the previous example, 10000 runs on such processors of the operation in Equation 1.1 will return a similar variability to what was demonstrated with the permutations in Equation 1.7 and Figure 1.4.

2.2 MOTIVATIONS

The body of contributions added by worldwide research teams through reproducibility support tools is to be acclaimed. As of January 3rd, 2018, we have listed 40 of them. Despite, the possible improvements proper to each tool, our aim in this thesis is not to provide another tool that will provide a different solution to the current ones. Instead, we ought to appropriately reason about how to best contribute to the existing efforts. Thus, we sincerely propose to inquire on the problem statements in the previous section 1.2. As such, we have formulated two observations and proposed solutions as hypothesis currently in experimentation:

O1. Can we design a system that will appropriately address P_1, P_2 and P_3?

Going through **P_1**, **P_2** and **P_3** in the previous section gives the feeling of similarity and dependency. In fact, the issue of collaboration expressed in **P_3** seems to be the fundamental glue. First, in **P_1**, the need for reaching out better can benefit from collaboration. Moreover, collaboration implies communication and the sharing of knowledge. Consequently, it will inherently drive awareness. Also, in a reverse manner, collaboration can benefit from more effective awareness driven schemes for reaching out. The more people are aware of the problem and the solutions, the more people will collaborate. Second, in **P_2**, the lack of standards or interoperability mechanisms limits significantly the scientists' freedom to use any tool of their liking. The reality is that every scientist will most likely not be introduced to the same tools at the same time. Thus, collaboration between scientists will take a toll, which will escalade to **P_1** as discussed before. In this thesis, we have indeed taken the challenge of building such as system.

O2. Can we craft a method that will allow to finally answer to P_4?

P_4 requires going beyond the scope of documenting the tools and scientific methods pipelines that contributed to a specific result. This explains why the current tools are deprived when facing numerical precision issues. Additionally, as mentioned earlier, our problems in **P_4** are the manifestation of the limitations of what today's computers can achieve. Thus, we venture in search of a solution.

2.3 CONCLUSION

In this chapter, we elaborate on four principal problems and our research motivations to address them. These problems have been identified as impeding scientific results reproducibility. Moreover, we have reasons to believe that they limit adoption of the tremendous efforts contributed by research teams in various institutions across the world. Beyond reproducibility, there is a growing concern in genuinely corroborating new scientific claims. The present thesis does not pretend to solve all the subsequent underlying issues. Instead, we have come to realize that a scheme for unifying the current contributions will be far more beneficial and impactful. As such we specifically propose to solve to the issues of the existing efforts.

Leaving this chapter with a clear view of what concerns us in the matter of advancing reproducibility, we take the reader onto the critical chapter of what is out there. The following chapter informs the reader on the current methods used to solve reproducibility issues. It shows the strength of those techniques and their limitations which will mostly reflect our main concerns expressed in the present chapter.

Chapter 3 – LITERATURE REVIEW

3.1 INTRODUCTION

It is clear now more than ever before, that the complexity and speed at which our technological shifts occur are impeding our ability to properly reproduce scientific results without specialized tools. However, there is a growing consensus that solving these problems requires the proper identification and recording of some key elements that contributed to these computational results [**Sandve et al. 2013**]. They are: the research environment, the experiment dependencies, the experiment inputs, the experiment executable and the experiment outputs. The following definitions express how these key elements were understood during this thesis based on the literature.

Definition 3.1. Environment means enough information about the system in which the simulation/experiment was run (hardware, operating system and compiler).

Definition 3.2. Dependencies represent all elements such as the libraries required by the simulation code/experimental design to be properly built and executed.

Definition 3.3. Inputs contain all the data ingested by the simulation/experiment to produce the expected outputs at the end of its execution.

Definition 3.4. Executable signifies enough information to retrieve/recover and execute the simulation/experiment (lab procedure, experimental design, source code, binary, execution command).

Definition 3.5. Outputs refer to all the data produced by the simulation/experiment during its execution.

The identification and documentation of these five elements in a way that fosters reproducibility is currently regarded as a gold standard in various efforts [**Stodden, et al. 2014**]. Currently, more journals are asking authors to publish more than the data they used to provide within traditional articles. As pointed out in [**Donoho 2010**], Dr. Claerbout slogan is most adequate here:

An article about computational Science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

In this chapter we would like to first, inform the reader on the current landscape of software that are being developed in support of scientific results reproducibility. The rich landscape of research and software in the first section is the cornerstone that motivated this thesis. Then, we guide the reader in a hierarchical and comprehensive exploration of the current landscape of platforms in the literature that attempt to solve or partially solve $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ or $\mathbf{P_4}$.

3.2 AN OVERVIEW OF REPRODUCIBILITY FOCUSED SOFTWARE

The current landscape of core research that focuses solely in the problem of reproducing a result is composed of three major categories. First, we have research that focuses in general purpose reproducible research. The two categories in this cluster are: Literate Programming ($\mathbf{C_1}$) and Execution Wrapping ($\mathbf{C_2}$). Despite their fundamental opposite philosophies, these two complementary categories focus on reaching a point where another scientist can successfully re-run a previously executed experiment to the point of getting a result. On the other end, the third category aims for an exact run to run reproducibility that yields identical results: Numerical Assessment ($\mathbf{C_3}$). The latter does not care about the recording of the previously defined five key elements. Instead, it focuses on methods that guarantee identical numerical reproducibility. In the following subsections we propose an overview of these three categories.

3.2.1 ON GENERAL-PURPOSE REPRODUCIBLE RESEARCH

For a scientist who has a hard time reproducing another scientist result, there is likely a feeling that this journey be memorable. More importantly, one can acutely recall the most burdening aspects. Intuitively, these can be aggregated in one single question that most corroborators ask: How can I create the equivalent conditions to run this experiment again? Most of the tricks, ideas and research pulled off to answer this question again and again lead to at least looking to simply run it again. At this point we don't even think about the result as we spend hours struggling to

install hardware specific drivers, setup libraries, find the right OS configurations and so on [Hothorn and Leisch 2011].

As such, general-purpose reproducible research attempts to provide solutions in two categories to answer this question. Thus, the methods presented here focus in capturing the five key elements listed in the introductory section of this chapter. In fact, they are thought to be enough to allow a reconstruction of the conditions behind the correct execution of previously run experiments. Moreover, with the tools that implement the methods presented in this section, the corroborating scientist can focus more in the actual corroboration of the result. We mean, seeking to know if the results of the new run are within acceptable domain specific error scales or why not identical to the original. Table 3.1 provides a non-exhaustive list of tools implementing these categories.

Disclaimer: *Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), the Université Clermont Auvergne (UCA), the Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS), nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.*

Table 3.1 Software supporting general-purpose reproducibility of scientific results

Tools	**C_1**	**C_2**	**M_{ethod}**	**Tools**	**C_1**	**C_2**	**M_{ethod}**
ActivePapers [**Hinsen 2015**]	✓	✓	ExP	PANDA [**Dolan-Gavitt et al. 2015**]	×	✓	PoW
Arvados [**Guthrie et al. 2015**]	✓	✓	WoM	Nextflow [**Di Tommaso et al. 2017**]	✓	×	WoM
Autosubmit [**Badia et al. 2017**]	✓	×	WoM	Nipype [**Gorgolewski et al. 2011**]	✓	✓	WoM
CDE [**Guo 2012**]	×	✓	InW	ReproZip [Chirigati et al. 2016]	×	✓	InW
Codalab	✓	✓	ExP	rkt [Wood 2017]	×	✓	PrW

Dask [**Rocklin et al. 2015**]	✓	×	WoM	runc [**Gantikow 2017**]	×	✓	PrW
Docker [**Turnbull 2014**]	×	✓	PrW	SHARE [**Gorp and Mazanek 2017**]	✓	×	ExP
Fireworks [**Jain et al. 2015**]	✓	×	WoM	Sumatra [**Davison 2014**]	×	✓	InW
Jupyter [**Perez and Granger 2007**]	✓	×	CoN	VCR [**Gavish and Donoho, 2011**]	✓	×	ExP
Kepler [Altintas et al. 2004]	✓	×	WoM	Vistrails [**Callahan et al. 2006**]	✓	×	ExP
OpenMole [**Reuillon et al. 2013**]	×	✓	WoM	NoWorkflow [**Pimentel et al. 2017**]	×	✓	InW

For every tool, Table 3.1 shows which one of the two categories is implemented. It also provides the primary method (presented in the following subsections) implemented by the tool: ExP (**Executable Paper**), CoN (**Computational Notebook**), WoM (**Workflow Manager**), PrW (**Pre-Execution Wrapping**), InW (**In-Execution Wrapping**) and PoW (**Post-Execution Wrapping**).

3.2.1.1 LITERATE PROGRAMMING

Introduced by Pr. Donald Knuth [**Knuth 1992**], the credo of literate programming is that most problems in computer Science such as software maintainability and results reproducibility come from the fact that we conventionally focus mainly in telling the computer what to do. Instead, we should be focusing more in embedding enough information to tell another human being what we want the computer to do. In fact, the way we tell computers what to do through programming languages and how computers understand them based on their CPU instruction sets and architecture continuously change. One well-known example among others is certainly file format obsolescence [**Rosenthal 2010**]. As such, literate programming is an adequate mixture of code and a guided explanation of what we intend to do with it as shown as following from **Inweb** (**http://inform7.com/sources/inweb** accessed July 17, 2018).

ϕ13. So here the head of one sequence is $T^{p_1}$and the head of another is T^{p_2}, so in the product we ought to see $(T^{p_1})^{s_1}.(T^{p_2})^{s_2} = T^{p_1 s_1 + p_2 s_2}$. But we don't enter terms that have cancelled out, that is, where $p_1 s_1 + p_2 s_2 = 0$.

⟨Both terms refer to the same base unit, so combine these into the result 13⟩ ≡

```
int p = p1*s1 + p2*s2;                                    combined power of t1 = t2
if (p != 0) {
   if (result ⟶  no_units_pairs == MAX_BASE_UNITS_IN_SEQUENCE)
      ⟨Trip a unit sequence overflow 15⟩;
   return⟶  unit_pairs[result⟶  no_unit_pairs].base_unit = t1;
   return⟶  unit_pairs[result⟶ no_unit_pairs++].power = p;
}
t1 = UNKNOWN; t2 = UNKNOWN;                          dispose of both terms as dealt with
```

ϕ14. Otherwise we copy. By copying the numerically lower term, we can be sure that it will never occur again in either sequence, so we can copy it straight into the results.

Thus, literate programming has the advantage that when problems occur, another scientist with the appropriate background will understand the embedded information. Hence, it will be possible to either fix the issues or migrate the software to a newer working environment.

In this section, we are referring to literate programming in a slightly larger sense than what was coined by Pr. Knuth in his WEB software [**Knuth 1984**]. The former focused on software source code literacy. Here by literate programming, we also propose to include metadata in support of today's complex software pipelines executions. In the following subsections we describe the three major methods that we have identified within the concept of literate programming.

Executable papers

More aligned with the original idea of Pr. Knuth, executable papers [**Strijkers et al. 2011**] are more than just publishable artifacts of their former academic version, now thought to be obsolete [**Somers 2018**] or otherwise inaccessible [**Taylor and Taylor 2018**]. Their main feature is that, the typically inserted results as tables, figures and graphs are the outputs or links to codes computed on the fly when rendering the paper. Moreover, any client reader can rerun these codes and therefore regenerate all the results in a more corroborative and interactive way than ever before. The evident advantage of the codes living alongside their scientific argumentations

is that they are the implementations of the latter and produce the results that support the claims advertised in the paper, all in one shot. While traditional academic papers can be read either on physical support or with software supporting their files formats (PDF, TEX, Word, etc.), executable papers require specialized software and services to be properly viewed or to run the codes. Such software and services can be identified as executable paper servers and clients usually designed as shown in Figure 3.1.

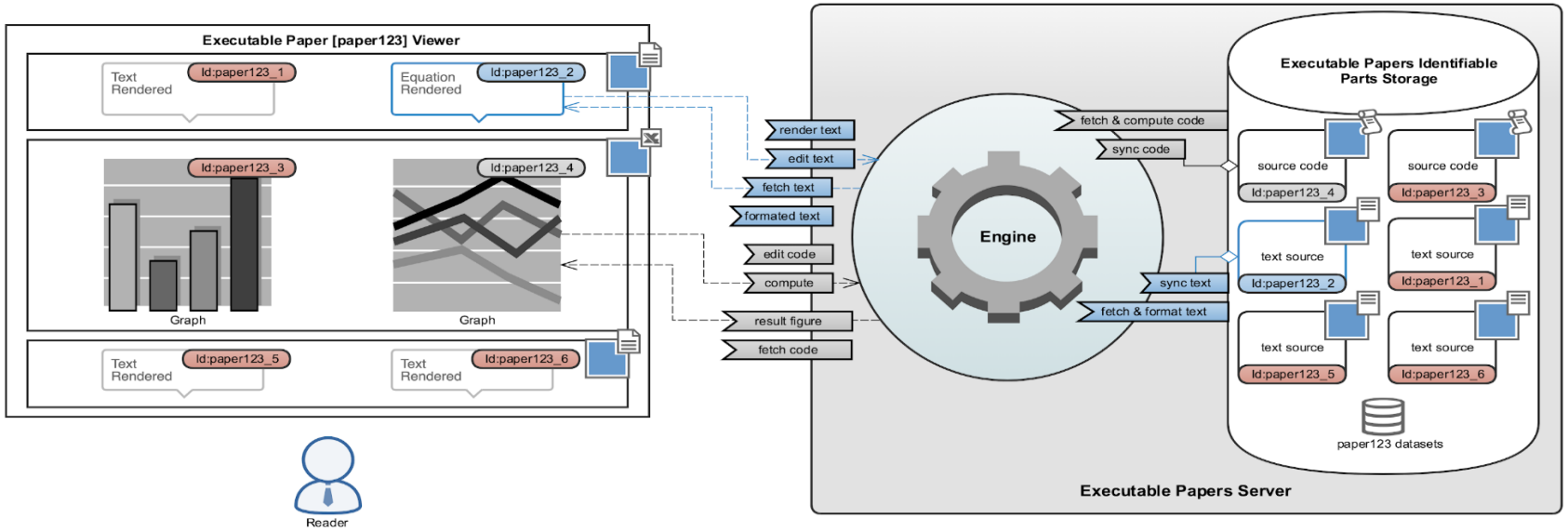

Figure 3.1 Generic architecture of an executable paper server: Inside the viewer, the reader can edit and re-compute the source of the any content being rendered.

From within a client viewer, a reader of an executable paper is viewing computed and rendered parts of the paper by an executable paper server engine. Typically, an executable paper is identifiable with a unique Id and is composed of computable parts also identifiable with some unique Ids. A computable part contains either a text script that can be executed to render formatted text or a source code that can be computed or a link to either of the formers. In the case of the source code, the result is typically rendered by a figure object. Hence, the reader can request the live re-computation of every parts of the paper. Moreover, in some cases, the reader can edit the paper parts sources. Thus, the embedded source code can be executed on a different dataset or even be altered during the viewing session. In the case of links, the reader is taken to a subsequent view (generally in a browser) in which they can perform the previously described actions. Table 2.1 provides five executable paper tools that we briefly describe in the following.

- **ActivePapers**. The research behind this tool mostly focuses on a file format for storing computation. The goal of this R&D work is to help scientists produce research that is more reliable and easily publishable. In ActivePapers' philosophy, data is more important than software. However, everything is again data in the tool's vision. Thus, the tool stores everything using HDF5 [**Folk et al. 2011**] as its underlying storage format. Therefore, datasets within an ActivePaper can be analyzed using any generic HDF5 viewing tool such as **HDFView** (**https://support.hdfgroup.org/products/java/hdfview** accessed July 17, 2018). Moreover, platforms such as figshare [**Kraker et al. 2015**] and **Zenodo** (**https://zenodo.org** accessed July 17, 2018) are great venues to publish and view ActivePapers.
- **Codalab**. It is an executable paper web server that provides sandboxed environments to researchers. Within these sandboxes, scientists can run their code on the data they uploaded. Thus, CodaLab makes it easy for collaborators to re-run the same code on the same data, to run the code on new data, or to even run the code with some modifications. To use Codalab, scientists must understand two major concepts. First, sandboxed environments are called bundles. They are immutable files/directories that represent the code, data, and results of an experimental pipeline and are submitted to CodaLab as data and/or code containers. The goal of the bundles is to enforce

reproducibility. Second, the concept of an executable paper is called worksheets in Codalab. Furthermore, they can be seen as lab notebooks and tutorials material. They are meant to organize and present an experimental pipeline in a comprehensible way. CodaLab's server engine is called in its jargon: the worker system. It manages the bundles and the worksheets created, modified and uploaded by the scientists. It is composed of three components. The first one is the core server. It's a REST [**Pautasso et al. 2008**] RPC [**Srinivasan 1995**] server responsible for processing requests from the client viewers (browser) and the worker. It also handles the collection and storage of metadata and all data in a database and on disk. The second component is the worker node. The CodaLab server engine is composed of a distributable set of worker nodes that can be run on machines with available compute resources. It executes scientists' commands within bundles as Docker containers. The last component is the bundle manager. It's a process that continuously fetch bundles in the database and schedules them to run on workers.

- **SHARE**. It is a product of the challenges faced by the Transformation Tool Contest (TTC) which encourages submission of software even at prototype phase. Such challenges scope difficulties to install or consistently configure the software. Additionally, there is no guarantee that the current version of the software will be available in the future. TTC is an event that focuses on a yearly basis evaluation and dissemination of advanced transformation techniques and related software. As such, SHARE was developed as a solution to provide environments in which all software and related data are installed, configured properly and ready for evaluation. Within SHARE web portal, scientists can create, share and access environment remotely as virtual machines from researcher's papers. Thus, results can be published alongside the links to the computation that produced them in SHARE. When clicked, the reader is taken to a live virtual machine session provided by SHARE in which they can re-run the computation as published or modify them at will. The Technische Universiteit Eindhoven (TU/e) University of Technology has been providing a free academic instance of SHARE.

- **VCR**. The Verifiable Computational Result is the outcome of research in which computations, the data they ingest and the results they produce get assigned Verifiable

Result Identifiers (VRIs). It is designed to deliver the same working environment as what scientists are currently used to. As such, scientists using VCR produce computational scripts and word processor files that are very much like those they work on today with a few subtle changes. In fact, these scripts generate a stream of verifiable results that are the same tables, figures, charts and datasets the scientist would typically produce but assigned a VRI and stored in a VCR repository. Thus, within the community, collaboration involves sharing those VRIs. Moreover, when included in papers, the reader is taken to a web portal in which they can locate, browse and when appropriate re-run the computations that generated the results. To reference data and source code, the scientist need to commit them along with the libraries used to a VCR server instance. Following the proper order, the data must be submitted first to retrieve VRIs that can be used within the source code to access them. Then, source code can be uploaded to retrieve a VRI that can be referenced to run it. After the execution of the source code, the VCR server sends back an email to the scientist, which contains VRIs of all the results produced during the execution.

- **Vistrails**. It's a tool whose research aims at integrating data acquisition, derivation, analysis, and visualization as executable components. Thus, these components will facilitate the generation and sharing of repeatable results. Compared to its pairs, Vistrails certainly differentiate itself through its mechanism for capturing metadata and provenance information from source code and libraries. It engages with authors, reviewers, publishers and readers throughout the paper life cycle. It keeps track of the computations, the data and the executed parameters while scientists do their research. Later, the resulting data, plots or visualizations can be referenced in the paper in a similar way as done with VCR.

Computational notebooks

Like executable papers, computational notebooks [**Rule et al. 2018**] are computed and generated by notebook servers. Despite their similarities in principle, notebooks and executable papers are quite different in their purpose. First, while an executable paper aims at enhancing the former academical standard research paper, notebooks intend to instead make the whole process of

research investigation and teaching easier. As such, a notebook is a reasoning interface like a digital whiteboard on which scientists can test their ideas, build methodical teaching materials and finally distribute them without the hassle of a review board. Therefore, most notebook tools support more programming languages and allow scientists to leverage and use most libraries available in those programming languages. Even though notebooks were not initially meant for academic publication, media giants such as **O'Reilly** (**https://www.oreilly.com/ideas/jupyter-at-oreilly** accessed July 17, 2018) are interestingly adopting notebook tools. In Table 2.1, Jupyter is certainly the most popular notebook tool available. The Jupyter notebook provides a web-based server allowing scientist to open sessions in which they can: develop, document, format text and execute source code. Thus, the formerly named IPython notebook contains two main components. The first component is a web server. It is a tool that allows interactive integration and execution of code, explanatory text scripts and their formatted outputs in notebooks from within a browser client to the server machine on which it is installed. The second component is the notebook. It is a document holding all the data, text scripts and source codes as inputs from the scientist; and all their outputs as internal objects, formatted texts, tables, graphs and images. Within a Jupyter notebook browser session, a research can first edit syntax highlighted source code or rich text [**Ovadia 2014**] as any modern text editor. Then run the code or rich text on the server machine from the browser and receive the results. And finally display the results of code and rich text (mathematical notation) computations as rich media representations through HTML, LaTeX, PNG, SVG, etc.

Jupyter and other notebook tools not listed in Table 3.1 such as **BeakerX** (**http://beakerx.com** accessed July 17, 2018), **Kajero** (**http://www.joelotter.com/kajero** accessed July 17, 2018) and **Zeppelin** (**https://zeppelin.apache.org** accessed July 17, 2018) are based on the idea of a lab notebook, brought to life in web browsers. Each notebook is a place for recording the written ideas, data, images, spreadsheets, diagrams, equations, and especially code, that one produces in the course of research. Scientists can analyze, visualize, and document data and Science, using multiple programming languages.

Workflow management

Unlike the executable papers and computational notebooks methods which focus on literate programming from within the source code, the workflow management method addresses the aspect of software execution. In fact, the modern in-silico experimental scientist investigation involves an ever more complex pipeline of tasks. Each task in the pipeline will typically ingest some inputs and produce outputs that are fed into the next tasks as inputs and so on until the last tasks are reached [Altintas et al. 2004]. Moreover, a task can be a simple function, a complete software or a webservice. Well-known job schedulers such as **Slurm** (**https://slurm.schedmd.com** accessed July 17, 2018), **TORQUE** (**http://www.adaptivecomputing.com/products/open-source/torque** accessed July 17, 2018) and **Grid Engine** (**http://www.univa.com/products** accessed July 17, 2018) are either being replaced or wrapped by new tools in order to provide a better literacy to the design, execution and management of current scientific pipelines. Thus, newer scheduling scripts embed more explanatory information detailing the pipelines, their purposes and how to launch them in a reproducible fashion.

Most workflow management tools implement literate programming by providing specialized languages or software libraries that are used by the scientists to craft their pipeline recipes. Figure 3.2 taken from Dask respectively showcase a Dask recipe and its resulting computational pipeline.

```python
import dask

@dask.delayed
def inc(x):
    return x + 1

@dask.delayed
def double(x):
    return x + 2

@dask.delayed
def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    a = inc(x)
    b = double(x)
    c = add(a, b)
    output.append(c)

total = dask.delayed(sum)(output)
```

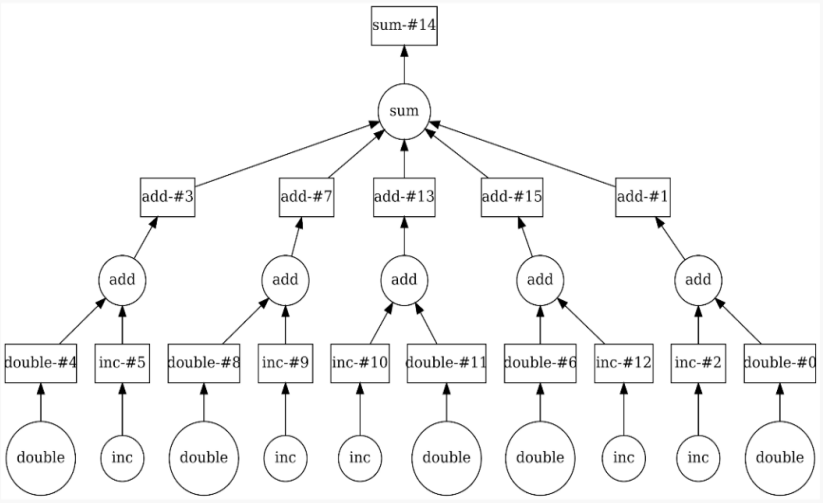

Figure 3.2 Computational workflow recipe and graph with Dask: This pipeline aims to sum the result of add of the results of inc and double for the five digits in data. The resulting graph seems less complicated when looked at with the recipe that generated it. Despite its complexity the graph visually details the chain of calls that produces the final result.

The resulting pipeline is a Direct Acyclic Graph (DAG) of three tasks (inc, double and add) composed to achieve a Single Instruction/Multiple Data (SIMD) execution paradigm. Here the single instruction is a graph branch involving a single instance of *inc*, *double* and *add* tasks used on a single value of data which can be interpreted in the case of **1** as *add*(*inc*(**1**), *double*(**1**)) and intermediary results double-#0, inc-#2 and add-#1.

In some ways, complex software executions today are done by source code that is also executed [**Amstutz et al. 2016**] and thus would benefit from literate programming. In the previous Dask example [**Rocklin et al. 2015**], while the source code can benefit from more commentary efforts, the generated graph provides enough literacy so that the idea can be implemented again with other tools and fixed in the advent of an error or Dask becoming obsolete in the future.

In Table 3.1, we list eight workflow-based tools used by scientists. We provide a brief description of each in the following.

- **Dask**. It's a workflow engine based on a novel specification that allows scientists to write their workflow as a Python program. With the use of objects such as Dicts, Tuples and Callables, the Dask specification allows a direct mapping of its workflows to their corresponding DAGs. As such when a Dask recipe is executed, the scientist can visualize the live execution of the tasks through a **Bokeh** (**https://bokeh.pydata.org** accessed July 17, 2018) interface. Hence, the scientist can have both the source code of the tasks and the recipe of their intricate execution in a single place with the ability to provide extra explanatory text. Moreover, the produced graph figures are additional literature explaining what was intended.
- **Kepler**. It's a platform for executing scientific workflows. It allows researchers to compose heterogenous software components written in different programming languages. Inside Kepler, a workflow is composed of components connected to each other and to data sources. As such scientists using Kepler can share and reuse data, workflows and components developed by the scientific community. Moreover, workflows can easily be executed locally or in a distributed fashion. Kepler inherits both its GUI and workflow system from **Ptolemy** (**https://ptolemy.berkeley.edu** accessed July 17, 2018).
- **Nipype**. Neuroimaging in Python is a software package that eases the development and integration of neuroimaging-based data analysis algorithms. These algorithms are represented as workflows along with their inputs and outputs described in an object-oriented fashion. Nipype gives Neuroscientists the capability to use a plug-in architecture to run workflows locally, on multi-core machines and remotely on clusters.
- **Fireworks**. It is a python workflow software that leverages the equivalence between Python Objects (Dict, List, Tuple, etc.) and JavaScript Object Notation (JSON) to deliver a direct mapping between the representation of its workflow objects and how they are stored. With a powerful query language, scientists can directly perform search operations and recombination of workflow parts (jobs) that can be intuitively executed by FireWorks. Similarly, to other workflow tools, FireWorks workflows can be executed locally in parallel or distributed across remote workers.

- **Arvados**. It is a workflow platform for data Science applications involving very large data sets. It is composed of two major parts. First, **Keep** (**https://dev.arvados.org/projects/arvados/wiki/Keep** accessed July 17, 2018) is a storage system for large files with an addressable content feature. Second, **Crunch** (**https://doc.arvados.org/user/tutorials/intro-crunch.html** accessed July 17, 2018) is a workflow engine based on container technology. Thus, scientists using Arvados, can manipulate flexible, scalable, versioned and reproducible workflows. Moreover, they will additionally be able to seamlessly access and manage large amounts of data.
- **Autosubmit**. It is an High-Performance Computing (HPC) utility that allows scientists to manage their workflows inside clusters, and Supercomputers remotely and mostly via SSH (Secure Shell). With Autosubmit, researchers can manage dependencies between computing jobs and leverage an HPC specifics agnostic layer that will not require code re-adaptation when sharing the workflow between clusters. Moreover, this tool allows provenance capture in a way that allows automatic retrials and the capability to rerun parts in case of a failure or a corruption.
- **OpenMole**. Although designed to give more control to researchers around their numerical models, this software provides more in terms of how to run the latter. OpenMole, models are designed as workflow nodes that can be run independently of the programming language and the type of input/output space. Furthermore, the software allows a seamless scaling feature across servers, clusters, grids and clouds.
- **Nextflow**. Similarly, to Arvados, this tool allows the construction of data-driven computational pipelines that are scalable and reproducible scientific workflows. Its similarity to Arvados comes from the fact that it also leverages container technology to bundle experiments. It differentiates itself from other tools due to its Domain Specific Language (DSL) which allows the implementation and deployment of complex parallel workflows in HPC infrastructures.

3.2.1.2 EXECUTION WRAPPING

An opposing philosophy to the one of literate programming considers that issues in corroborating scientific results come not from relying too much on computers but instead from the trust in the

human factor. Moreover, this philosophy focuses mainly in capturing the execution environment [**Davison 2016**]. Figure 3.3 shows the three execution wrapping methods that are implemented to put the trust on the computer by allowing it to automatically wrap its execution in a corroborative fashion. This category scopes methods that puts the blame on unguided manual operations and the memory of the scientist. Errors will be introduced through the lack or loss of precision in operations. Moreover, forgetting cannot be avoided especially when documenting key elements to ensure corroboration. Without a consensus, what one think not important to be documented might be important to another in the process of understanding. Thus, everything must be automated. The machine should be given the control to record and guide the scientist during the investigation in a corroborative fashion. Therefore, another machine will automatically be able to perform the same operations again from that record.

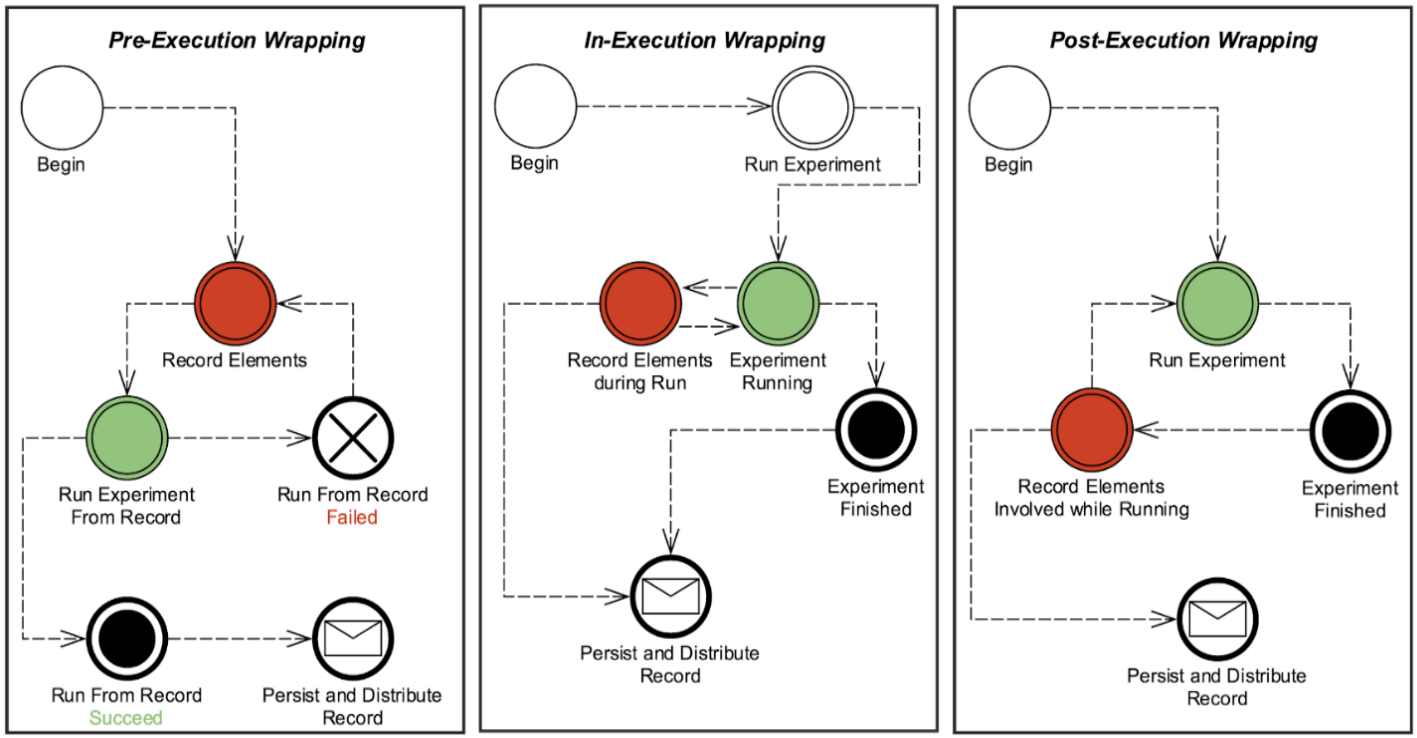

Figure 3.3 Diagrams of the three execution wrapping methods: These methods can be distinguished from the precedence of the wrapping/recording process shown here as the red state with regards to the actual execution represented here with the green state.

Pre-Execution wrapping

In execution wrapping, the present method involves a prior determination of the investigation key elements as shown in Figure 3.3 downmost diagram. This is referred to in [**Davison 2016**] by pre-emptive capture. Typically guided by a tool, the scientist can produce the record of what is needed to reconstruct the investigation. Such record is generally in a standard format that any tool aware of, can execute. The Open Container Initiative (**OCI** (**https://www.opencontainers.org** accessed July 17, 2018)) Image Format Specification is an appropriate example. When using pre-execution wrapping, scientists spend the major part of their time assessing the record. In fact, the record must reach a first successful run which demonstrates that the environment, the dependencies, the inputs and the executable are properly wrapped and produce the expected result. Pre-execution wrapping tools speculate on the fact that when the wrapping follows some guidelines, the resulting record or its equivalent build is guaranteed to always work with any other supporting tool. Thus, a scientist that wants to corroborate the experiment results, needs to use the same tool that produced the record or another tool that supports its format. According to **Github** (**https://github.com** accessed September 21, 2018), the top four pre-execution wrapping tools in Table 3.1 are: Docker, rkt, runC and Nextflow. The following subsections briefly review them.

- **Application Container Systems.** Differently from hypervisor virtualization which virtualizes the hardware layer to accommodate one or more independent machines, containers instead come on top of the Operating System (OS) kernel to split the user space into small sandboxes scoped for applications build, deployment and execution. They are run within the user space on top of an operating system's kernel. Hence, container virtualization is often referred to as operating system-level virtualization [**Morabito et al. 2015**]. Docker [**Turnbull 2014**] is a container system that allows scientists to capture key elements to enhance corroboration by following two guidelines. The first one involves the capture of the environment, the software, its dependencies and how to run it in a file named Dockerfile. With such a file, scientists can custom tailor the operating system and prepare the ideal self-contained environment in which their simulation can run. A Dockerfile gets built into a Docker container which in turn is persisted on the host and can

be launched as a guest. Moreover, scientists can generate lightweight images from their Docker containers. Thus, these images can be distributed to others directly or through the Docker Hub (https://hub.docker.com/ accessed September 29, 2018). The second guideline helps scientists custom design the execution requirements of their containers. This is expressed on a file named Docker-compose in which the scientist can express where inputs can be found and where outputs should be stored. Within such a file, scientists can also combine Docker containers to produce a complex combination of well contained scientific codes. As soon as scientists can wrap their simulations inside Docker containers and provide a Docker-compose file to run them, they have done most of the work. Then, collaborators need only to install the Docker tools suite to be able to reproduce their results. Similar tools such as rkt [**Wood 2017**] and runC [**Gantikow 2017**] are container engines like Docker that come with equivalent mechanisms to pre-wrap scientists simulation codes in containers before executing them. Yet these two tools are slightly different from Docker. For example, rkt, introduced by **CoreOS** (**https://coreos.com** accessed July 17, 2018) and naming its containers, *pods*, has a design philosophy that is similar to **Kubernetes** (**https://kubernetes.io** accessed July 17, 2018). It does not rely on an intermediate daemon. Instead, a pod is executable directly and is always in non-root state. Moreover, to create the image of a simulation, scientists must use **acbuild** (**https://github.com/containers/build** accessed July 17, 2018) which is a command line utility to build and modify App Container Images (ACIs), the container image format defined in the App Container (appc) **spec** (**https://github.com/appc/spec** accessed July 17, 2018). As such, rkt's pod images are inherently following the Open Container Initiative consensus and are therefore compatible with other container engines images within the Initiative. Also, in order to use runC, scientists must have their containers in the format of an OCI bundle. If Docker is installed, scientists can use its export method to acquire a root filesystem from an existing Docker container. After a root filesystem is populated, they can generate a spec in the format of a config.json file inside their bundle. runC provides a spec command to generate a base template spec that can then be edited. From an OCI bundle, runC offers two ways of running them. The simplest

way is to call the run parameter of runC inside the bundle, which will allow it to create, start and delete the container after it exits. The second way allows the user to custom tailor the internals of the bundle by modifying the generated spec and accessing the lifecycle of the bundle, giving the user more power over how the container is created and managed while it is running.

- **Nextflow.** It is a platform that helps scientists run reproducible scientific workflows in a scalable fashion. It does so by enabling the usage of a Domain Specific Language (DSL) to create workflows on clouds and clusters. Moreover, every task within the workflow is wrapped inside a container. In fact, Nextflow supports Docker and **Singularity** (**https://singularity.lbl.gov** accessed July 17, 2018) as container engines. With the use of GitHub, it gives scientists the power to write self-contained and versioned pipelines that seamlessly reproduce any configuration. Nextflow is a pre-wrapping execution tool that is designed to serve scientists in large clusters. As such it relies on the famous **LSF** (**https://www.ibm.com/support/knowledgecenter/en/SSETD4/product_welcome_platform_lsf.html** accessed July 17, 2018) **SLURM** (**https://slurm.schedmd.com** accessed July 17, 2018) **PBS** and **HTCondor** (**https://research.cs.wisc.edu/htcondor** accessed July 17, 2018) as batch schedulers. Compared to container engine tools that can be used by a scientist alone, Nextflow is designed for scientific infrastructures that intend to enforce reproducibility in all research projects through a pre-execution wrapping method.

Intra-Execution wrapping

This method (shown in Figure 3.3 middlemost diagram) exercises a wrapping of the execution in two steps and is referred to in [**Davison 2016**] as run-time capture. First, during the computation it typically intercepts the execution calls to the investigation environment resources. It does so by either taking care of the execution process itself or by generally watching everything on the system. There are two possibilities. In the first one with a computational experiment, the tool implementing this method act as a parent process that handles the execution of the experiment. In the second one, with a computational experiment, the tool implementing this method act like a watching daemon that listens for activity in the system. Second, at the end of the execution, the tool automatically or by directing the scientist, produces the record of the experiment

execution. Thus, differently from the previous method which mostly involves container technology [**Gantikow 2017**], here, the execution is firstly done outside of the record. In fact, the record is generated from the first successful execution. Then, to corroborate the experiment, another scientist need not to worry more than having a tool that supports the record format and structure. The following present in-execution wrapping tools listed in Table 3.1.

- **Sumatra.** It's an in-execution wrapping tool that captures data from the execution context of the scientific software that it launches. It does so in three parts. The first part involves the use of version control to enforce the proper management of scientific source code and the continuous mapping of an executable to it source code version. The second part focuses in the snapshot of the inputs, outputs of the execution and the capture of metadata about the hardware, OS and libraries versions. The last phase enables the storage of Sumatra records in the file system and a database in a way that enables their versioning. The software comes with a web application that allows a visual interaction with the records. **Sumatra has indeed profoundly inspired various aspect of this thesis.**
- **CDE.** It is an in-execution wrapping tool that uses **ptrace** (**http://man7.org/linux/man-pages/man2/ptrace.2.html** accessed July 17, 2018) to automatically intercept scientific software execution calls to the system. Then, it stores all the files involved in the system calls following a hierarchy that allows them to be returned through a reverse use of ptrace again in future system calls during re-executions. CDE persists its records as a compressed file that can easily be distributed. Moreover, the re-run of the computation does not require any installation (except for CDE), configuration or root permissions.
- **ReproZip.** It also uses ptrace to automatically trace system calls issued by a scientific software. Yet, differently than CDE, this trace is used to create a reproducible package that contains both metadata and copies of the files accessed during computation. ReproZip comes with a secondary command line tool named ReproUnzip. This tool is responsible for unpacking ReproZip packages into an executable form which can be customized by scientists. ReproZip is different than both Sumatra and CDE as it produces both metadata and a full snapshot of the files used during the execution. As such while

the scientist can read the metadata for more clue, ReproUnzip is able to rerun the resulting packages or turn them into Docker images that can be executed as containers.

- **NoWorkflow.** It's an in-execution wrapping tool that enables the collection of scientific code execution provenance at different levels of granularity. It does so by identifying three forms of provenance. First, a definition provenance that represents scripts and their contents: functions, arguments, calls and static data. Second, a deployment provenance that represents the operating system, the environment variables and the library on which the code depends. Finally, an execution provenance that traces the execution of the code. Fully leveraging the Python programming language, this tool consequently allows a novel provenance collection of scientific code execution that enables a better reproducibility.

Post-Execution wrapping

Challenging in principle, this upmost method (in Figure 3.3) is barely implemented. Yet, it does not have less merit than the others to be listed here. Similarly, to Pre-Execution wrapping, this method is based on virtualization. However, instead of aiming to preserve the software stack only (from OS layer) as the former, Post-Execution wrapping focuses in preserving a hardware stack. Thus, it is a virtualization in which the code from another architecture is translated to the host one [**Rosenthal 2015**]. Tools implementing this method will typically have a way to emulate the entire investigation system. Thus, the scientist only needs to be able to execute the experiment on a system emulated by the tool. A record of the inputs, the dependencies, the executable and a clear specification of the system is sufficient. The novelty in tools implementing post-execution wrapping is that independently from the system where computations are being run, the tool will emulate the proper system on top of any host system. Thus, it will provide the subsystem needed by scientists to compute their experiments. This means that a executable built on a PowerPC on an early Apple OS will still be able to be run on a more recent MacBook Pro with an Intel i7 CPU or any other recent Personal Computer (PC) hardware architecture. We recommend the user to review Table 3.1 to recall the tools implementing this method and the previously presented ones.

PANDA is a post-execution wrapping tool that is based on the QEMU [**Bellard 2005**] emulation system which supports thirteen different CPU architectures. As such it allows the record and

replay of executions while enabling deep dynamic analysis. Since PANDA intervenes at the machine code level, the use of QEMU allows the full repeatability of replays.

However, record and replay performance in PANDA are known to be currently fairly slow. In the example provided in [Dolan-Gavitt et al. 2015] with **gzip-1.2.4** (**https://ftp.gnu.org/gnu/gzip** accessed July 17, 2018), PANDA itself is about 5% slower than QEMU 2.1.0. Recording incurs a slowdown of almost 2x, and replay adds about another factor of 2. So, replay is almost 4x slower than standard QEMU. This may not seem slow, but replay is noninteractive, and, in many cases, analysis plugins incur much larger overheads of 10-100x and so the replay slowdown is insignificant.

3.2.2 NUMERICAL ASSESSMENT

Despite the popularity of the two previously presented categories, we also provide literature demonstrating first hand that the recording of the "elucidated" key elements that contributed to a computation result is not enough to achieve reproducibility in some cases. In fact, no matter the methods of recording an experiment for reproducibility, when it comes to running it again, variations can occur in the results for other reasons, such as:

- **Unstable experiment by design.** The result of the experiment may be based on factors that may change from a run to another. As such, if not designed in purpose with care, reproducing a result becomes a stochastic event that reduces the odds of corroborating it.
- **Uncontrolled hardware execution decisions.** With our technological changes, hardware and operating systems might take different execution decisions as they are improved, based on various environmental conditions. Thus, reproducing a result may be as hard as recreating the entire hardware and operating system states at the various stages that concurred in producing the result. Certainly, this is not a trivial task.
- **Uncontrolled hardware flaws.** In june 2017, a new flaw was discovered impacting Intel HyperThreading on Skylake and Kaby Lake-based processors (Intel 6th and 7th Generation). All types of operating systems are affected as specified by a recent Intel

errata documentation. This documentation explains that under some complex micro-architectural conditions, short loops of less than 64 instructions that use AH, BH, CH or DH registers, as well as their corresponding wider register (e.g. RAX, EAX or AX for AH), can cause unpredictable system behavior sometimes leading to crashes and potential data loss.

During this thesis we have also explored this complementary aspect in the scope of computational Science. Hence, we provide here an overview of the third category (**C_3**) used to complement the two others in bettering the corroboration of scientific results.

A numerical result is an intricate combination of the arithmetical operations results evaluated by the CPU during the execution. None of the tools implementing the methods in the previously presented categories currently preserve any trace of that information except within the outputs. Yet, outputs usually provide a limited view into how they came to be. Or we wouldn't be doing any of this. Would we? Only final numerical values are typically the one contained it the result. Thus, identifying the origin of a variation is not a trivial task. Out of order execution is one of the modern optimizations provided by CPU constructors that helps speed up computations. Yet, as addressed by Intel [**Zitzlsberger 2014**], its impact on floating point operations and the level of difficulty involved in debugging such a situation is unprecedented.

There are currently two methods that are widely used to reduce numerical irreproducibility. These methods are in essence, approximations that when finely tuned [**Hill 2015**] yield interestingly precise results yet to be considered at a certain order of magnitude. The following subsections provide a brief overview of these methods.

3.2.2.1 Interval arithmetic

The approximation of PI by Archimedes $223/71 < \pi < 22/7$ may well be the de facto "hello world" introductive example to Interval arithmetic. By avoiding direct evaluation of numerical values, this method focuses instead in computing the upper and lower bounds of all numerical variables inside the simulation inputs and consequently its outputs. Implemented first as a software package in 1976 at the University of Karlsruhe, this method was part of the project to develop

Extensions for Scientific Computation (XSC) for various programming languages such as C++, Fortran and Pascal [**Bohlender et al. 1993**].

While this method can be used to reduce numerical irreproducibility, it is currently known to suffer from side effects of ending in unrealistic intervals of magnified uncertainties. Thus, the craft of interval algorithms must be practiced with precautions [**Revol and Theveny 2014**]. Moreover, well-established numerical packages should be preferred among others.

Since the approval of Standard 1788-2015 by IEEE, there have been a rising interest [**Heimlich 2015**] in implementing this method. Libraries such as C++ libieeep1788 [**Nehmeier 2014**] joined **GNU Octave** (**https://www.gnu.org/software/octave** accessed July 17, 2018) to vulgarize Interval arithmetic and its use in computational Science.

3.2.2.2 Uncertainty quantification

It is currently a challenging prospect to evaluate the degree of confidence in numerical precision since the same object of the research itself (physical processes) is yet to be fully captured by computer models. Uncertainty Quantification (UQ) regroups a set of rigorous methods that enables the attenuation of variations in scientific results. UQ is a trending field that has been yielding promising results in a variety of engineering domains.

Statistical techniques can be directly applied to simulations [**Raychaudhuri 2008**] as done in the following UQ methods to quantify and reduce the effect of uncertainty in varying results.

Stochastic design

Mostly known for its applications in nuclear physics and financial markets, it is now used in computational Science. Moreover, it is appropriate when the solution to a problem can be the aggregation of many smaller solutions with as many parameters as possible. The approximation nature of this technique relies on two factors. The first one is the quality of the domain of the randomly picked input values. And the second is the number of results from as many different randomly picked values as possible to cover the input domains. Known to be the most widely used category, the Monte Carlo method aims at increasing the precision while reducing the sources of variability within the finally aggregated result. However, due to the importance of

quality in the distribution of the random numbers, it is consequently critical that the quality of the generators used be enforced by sound parallelization techniques of those generators to guarantee reliable results [**Hill et al 2013**]. Thus, thorough method such as the one proposed in [**Hill 2015**], is needed to guide scientists into mastering the reproducibility of stochastic simulations results when parallelized.

Stratified sampling

The present method is recommended when inputs parameters domains can be exhaustively partitioned into disjoint subgroups. It is a variance reduction method that differentiate itself by limiting the population parameters for groups within the population. Additionally, it enables more manageable measurements and guarantees that measurements within a stratum have lower standard deviations. Hence, a correlated smaller error in estimation is to be expected [**Bucher and Bourgund 1987**].

Latin hypercube sampling

Co-authored at separate times by Eglajs, McKay and Ronald L. Iman (1977, 1979, 1981), it is a statistical method that generates a sample of plausible collections of parameter values from a multidimensional distribution. Thus, it enables inputs parameters values selection with the aim to yield output values that are the most accurate [**Nishimura and Matsumoto 1998**].

Surface method

Introduced by G. E. P. Box and K. B. Wilson in 1951, this method allows the study of correlation between several explanatory and response variables. It enables the recasting of mathematical models of physical processes as stochastic Partial Differential Equations (PDEs) in order to solve them using deterministic methods [**Barth 2011**]. As such, response surface is an approximation method that can be used on sequences for the design of experiments that will yield optimal results (low variability impact).

3.2.3 DISCUSSION

The presented categories remain the foundation of substantial contributions in terms of research and software that principally support scientific results corroborations. However, they present some limitations, do not address or partially address the four problems stated in this thesis.

In the case of numerical assessment, the current methods are still approximations that either require highly technical knowledge of random numbers generators and adequate parallel programming or a unique expertise in domains such as interval arithmetic. In the case of numerical precision due to out of order execution, the current methods specifically fail at providing a robust solution in run to run numerical exactitude.

In the case of the methods in the two general-purpose reproducible research categories, the fundamental problem of interoperability is left genuinely unsolved. In fact, independently from the categories and the methods used, most general-purpose reproducible research softwares produces one of three types of reproducible artifacts structures defined as following.

Definition 3.6. A **metadata-based** structure involves artifacts that mostly record information about the experiment in Text using languages and file formats like: eXtensible Markup Language (XML), JSON, Yet Another Markup Language (YAML), etc.

Definition 3.7. A **snapshot-based** structure contains copies of all the raw files and elements that were involved in the execution usually stored in a hierarchical manner within a compressed file. It contains automated guidelines that can be used to rebuild the environment and re-execute the experiment.

Definition 3.8. A **hybrid-based** structure contains both record about the experiment in a form of metadata and also as a **snapshot** of all the involved elements during the computation.

While each of these structure types have their advantages, they also present some challenges. The following Table 3.2 shows criteria used to qualitatively compare the three artifact structures types.

Table 3.2 Qualitative criteria for comparing the three artifacts structure

		Meaning	Values
H	**Human readable**	How easy is it for a person to read and understand the artifact content?	**U**nreadable, **R**eadable and **V**ery readable
O	**Likeliness of obsolescence**	How likely is it for the artifact structure to become obsolete?	**V**ery likely, **L**ikely and **U**nlikely
R	**Ease to reproduce**	How easy is it to go from the artifact to an actual reproduction of the result?	**H**ard, **F**air and **E**asy
S	**Ease to share**	How easy is it to share the artifact to other scientists and communicate around them?	**H**ard, **F**air and **E**asy
P	**Provenance content**	How rich is the artifact in terms of pedigree information about the result and its originators?	**P**oor, **R**ich and **V**ery rich
V	**Volume**	How big in average are the artifacts sizes?	**V**ery small, **S**mall and **B**ig

Each of these criteria embodies qualities that are important in corroborating results. We provide a meaning and their various values.

The previous table is then referenced in the following Table 3.3 which provides the comparison and the methods whose implementations tend to produce these structures.

Table 3.3 Comparing artifacts structures and implemented methods

	H	O	R	S	P	V	Methods
Metadata	Very readable	Unlikely	Fair	Easy	Very rich	Very small	Executable Papers, Notebooks, Workflow Management, In-Execution Wrapping.
Snapshot	Unreadable	Very unlikely	Easy	Fair	Poor	Big	Pre-Execution Wrapping, In-Execution Wrapping, Post-Execution Wrapping.

Hybrid	Readable	Likely	Easy	Fair	Rich	Big	Executable Papers, Workflow Management, Pre-Execution Wrapping, In-Execution Wrapping.

In Table 3.3 we would like to specifically communicate to the reader, that depending on the way the artifact is structured, the quality in terms of general usefulness for corroboration in a large sense may be impacted. Additionally, we augment the table by linking methods to structures types based on what the average tool that implements them produces.

Although, metadata-based structure is ideal for the understanding of others, sharing, unlikely to become obsolete because it is usually textual and cause a very low memory print, it is not trivial to reproduce research results from it. In the contrary, while snapshot-based artifacts are trivial to reproduce, they are significantly heavier in memory, human unreadable, can come in ad-hoc unstandardized forms which renders them highly obsolescent and hard to share. Furthermore, the hybrid structure while taking the better parts of the others suffers from its volume and is consequently hard to share.

In addition to the limitation of each structure, we must stress to the reader that tools producing the same structure types do not mean same structure contents. In fact, based on the overview done during this thesis, few to a very limited number of the listed software produces similar content despite following the same structure. These great factors clearly point that none of the presented categories address $\mathbf{P_2}$. Moreover, $\mathbf{C_3}$ does not still properly address $\mathbf{P_4}$ in our opinion. And it is clear that due to their general-purpose nature as explained in this chapter, $\mathbf{C_1}$ and $\mathbf{C_2}$ do not address $\mathbf{P_4}$ at all.

While having a Web interface and being collaborative can be byproducts in the implementation of the previously listed software, we argue that these are not their primary features. As such, in our opinion, neither $\mathbf{P_1}$ nor $\mathbf{P_3}$ are properly approached by any of the current software listed in Table 3.1.

In the following section, we propose to take the reader in an overview of aspects within the current reproducible research platforms that attempt or could attempt to provide partial solutions to the problems of interest of this thesis.

3.3 LANDSCAPE OF REPRODUCIBLE RESEARCH PLATFORMS

Before embarking in this review section, it is appropriate to inform the reader that at the moment of writing this thesis, there is no other reproducible research software and services that address $\mathbf{P_4}$ but the few ones cited in this chapter. As such, we will be overviewing the rest of the literature with regards to what they contribute in the episteme of solutions for $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$. Furthermore, it is important for the reader to grasp that the literature work presented here goes beyond the problem of reproducibility itself. It assumes that at least one of the previously described methods is implemented. We instead consider the overall bigger picture of democratizing these implementations and providing them sustainably. During this thesis we have identified three hierarchically dependent concepts leveraged to achieve this ultimate goal. There are three core properties aimed for by reproducible research platforms: Web transformation (**T**), Collaboration schemes (**C**) and Workflow models (**W**). We infer that research focusing in these concepts are attempting to respectively solve directly or indirectly $\mathbf{P_1}$, $\mathbf{P_1}$&$\mathbf{P_3}$ and $\mathbf{P_1}$&$\mathbf{P_3}$&$\mathbf{P_2}$. To be more specific, these concepts will be presented here in the order at which one includes its predecessors. For example, Web transformation will be a previous subsection to Collaboration schemes. By that we imply that the presented research and platforms within the Collaboration schemes concept presuppose a Web transformation aspect.

3.3.1 Web transformation (T)

It has only been 28 years that the first Web browser was released. Yet, applications of the World Wide Web have tremendously proliferated and still are to great lengths. Now, we have Web servers all over the world serving various content. Moreover, as of June 15, 2018, we have more than **4.5 billion indexed Web pages** (**http://www.worldwidewebsize.com** accessed July 17, 2018). The Web has transformed the way we access information and the way we inform others all over the planet [Kling 1991]. It has been a trending venue for new concepts such as digital marketing [**Stephen 2016**] and the renewal of others such as advertisement [**Ratliff and**

Rubinfeld 2010]. Today, Internet and the Web are fundamental parts of our day to day life so much that it is being sought to be considered as a **basic human right** [**Best 2004**].

While the possible applications of the Web are still limitless, all the current ones are still based on its original feature: interlinked Uniform Resource Locators (URLs). Moreover, despite the fact that anyone today can setup a website in minutes and be ready to reach out, the Web has grown in complexity. As such, low level applications for indexing the content of the Web and their portals are the first places users go to find new content [**Introna and Nissenbaum 2000**]. The rush in optimizing this mechanism and the marketing battle of having one's URL presented first in a query have gave an economic viability to big Corporates such as Google and Yahoo. To put this in numbers, Google famous search engine processes over **3.5 billion requests per day** (**http://www.internetlivestats.com/google-search-statistics** accessed July 17, 2018). Figure 3.4 shows In-platform full and Off-platform construction of the artifacts.

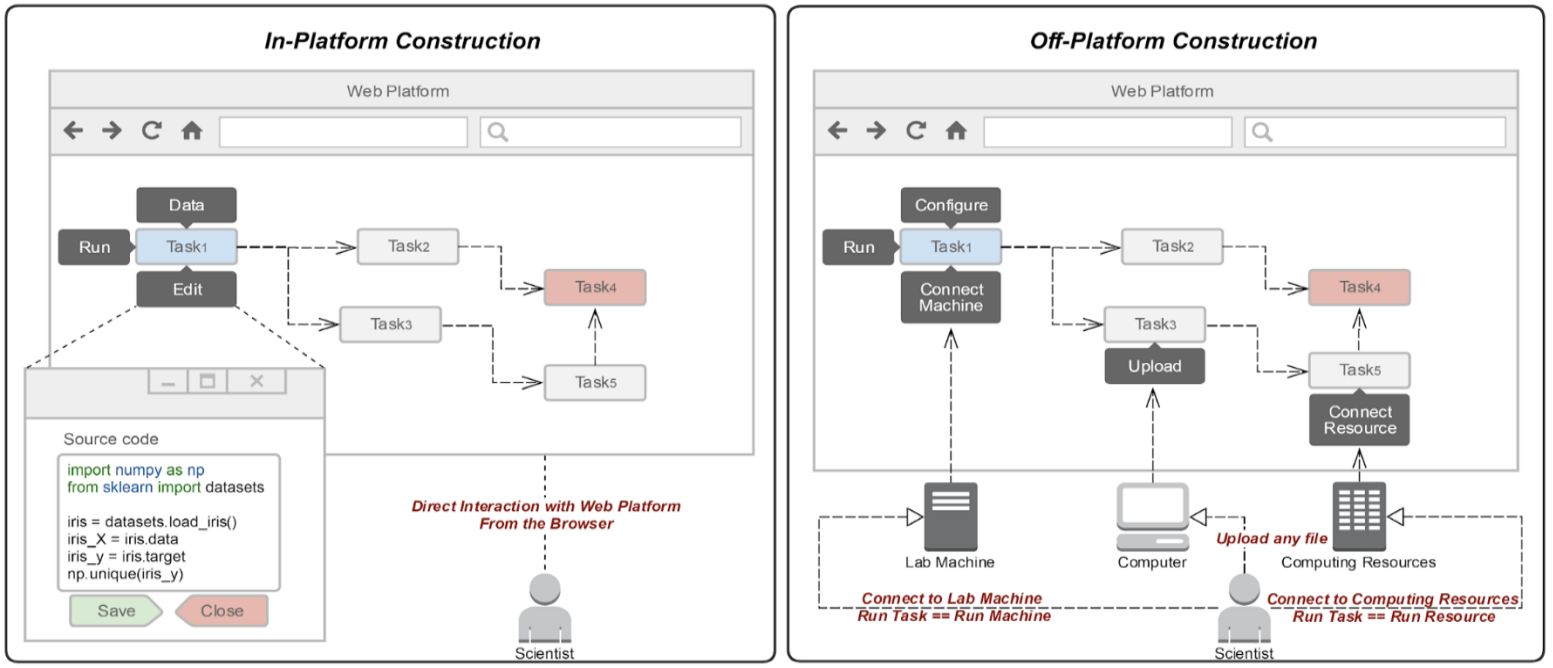

Figure 3.4 In-platform and off-platform artifact creation mechanisms demonstration: For in-platform all interactions are contained within the platform while for off-platform interactions may leave the platform to reach for a lab machine run, a computer upload or a computing resource execution.

In fact, we have identified two major categories used by web platforms to allow scientists to construct reproducible artifacts. An In-platform full construction of reproducible artifacts is a core feature of reproducible research Web platforms that provides a complete set of operations that allows scientists to perform their entire experiment from any supported Web browser. In computational Science, most implementations take the form of a literate programming-based Notebook method.

Alternatively, in the case of an Off-platform construction of reproducible artifacts, the platform will provide a guided mechanism to the scientist to construct the artifacts from externally generated content. As such, metadata will be inputted by the scientist or an authorized machine. Moreover, in computational Science, the scientist will be able to upload files as code, executable, dependencies, results, etc. Also, in experimental Science, an authorized machine will be able to communicate with the platform to contribute in the construction of the artifact in a meaningful way.

While the current use of the Web to deliver reproducibility capabilities and to reach out to scientists addresses $\mathbf{P_1}$, we argue that it still does so partially. The Web offers other features that may be leveraged. A few of those are: Multiple private/public launch configuration and a subsequent federation capability [**Gorelik 2013**]. Due to security, policy and management requirements, institutions across cities, states and countries prefer running their own web platforms in a segregated fashion with well-defined boundaries for the access and the sharing of information. As such, to ease adoption from scientists all over the world and consequently reach the broadest audience, Web platforms must be implemented in a way that:

- Allows anyone to launch an instance.
- Allows instances to be launched in private networks for restricted access.
- Allows instances to switch from private to public networks.

Moreover, based on access agreements across institutions, platforms instances should be connectable to each other to form a bigger federated entity [**Rubin 2015**]. Hence, scientists can search and collaborate across instances.

For these additional important capabilities, we infer $\mathbf{P_1}$ to be currently partially solved by the majority of existing platforms.

3.3.2 Collaboration Schemes (C)

As a core compartment of Science itself and a sparkling ingredient of its success, collaboration is the second feature implemented by most reproducible research platforms after enabling Web access. Moreover, reproducibility is not only meant for the original author. In fact, when we bring in the aspect of collaboration, we inherently [Lomas et al. 2008] mean involving another person to assess the reproducibility of results. Additionally, this concept is trending across all types of web platforms. It is specially with the applications in Social Media platforms (Google, Facebook, Twitter, LinkedIn, etc.). Consequently, a platform lacking collaboration appears as very limited and serving as a one-way publishing portal. Collaboration enables the bidirectional exchange which fosters more applicative aspects of the Web [den Exter et al. 2012].

We have identified two collaboration modes implemented within the current Web platforms: Segregated and Unsegregated collaboration.

Unsegregated collaboration is the simplest, easiest and less sophisticated mode that is typically managed by the scientists owning the artifacts. In such a mode, scientists can collaborate on reproducible artifacts through a privacy setting change. Artifacts are either set as private or public. When an artifact is private, it is not publicly visible to others. At this point only, the creator of the artifact can view and edit it. However, once made public, an artifact becomes visible and interactive to others. Thus, other scientists can view the artifact, download it and interact with the author and pairs on its web page by uploading content and sharing their thoughts typically through comments.

Inspired mostly from social media platforms, segregated collaboration involves a more sophisticated, finer grain interaction feature between scientists. In fact, one known limitation of unsegregated collaboration is that once shared, the scientist cannot filter who interacts with him or her on the matter of the shared artifact. It is either open to everyone or not at all. Alternatively, segregation allows more. Platform managers and in some case, users can create groups of

scientists and allow a fine grain sets of possible actions for each group. For example, a segregation scheme involving three groups could be as following:

- public: scientists in this group can only view shared artifacts.
- collaborators: scientists in this group can provide comments and upload files.
- team: scientists in this group can edit the artifact and provide direct modifications.

Implementing the latter mode is often done by the platform administrator.

Clearly, these two modes of collaboration and the platforms implementing them address **P_3**. Yet, they do so again only but partially. To be fully collaborative, a Web platform must provide mechanisms to accommodate other reproducible research platforms and tools. Even though most collaborative web platforms expose an Application Programming Interface (API), it is still an adoption barrier that is not exempted of burden. Also, sometimes as we will show later in this manuscript, most of the work must be done in the other tools and platform to allow accommodation. As such, we press our concern to the reader that **P_3** is not fully resolved here because most platforms provide this feature to their direct users. Moreover, due to **P_2** not being solved here we end up in a situation where scientists must duplicate their reproducibility efforts across platforms to be able to collaborate with larger audiences. To provide an example, users in a platform A will find a hard time to collaborate with users in another platform B if these two are not integrated to each other. Thus, scientists must learn to play with their respective APIs to at least have their artifacts synchronized between the two platforms. Such a work is not trivial and significantly impairs collaboration. Figure 3.5 provides a graphical expression of the two modes described in this section.

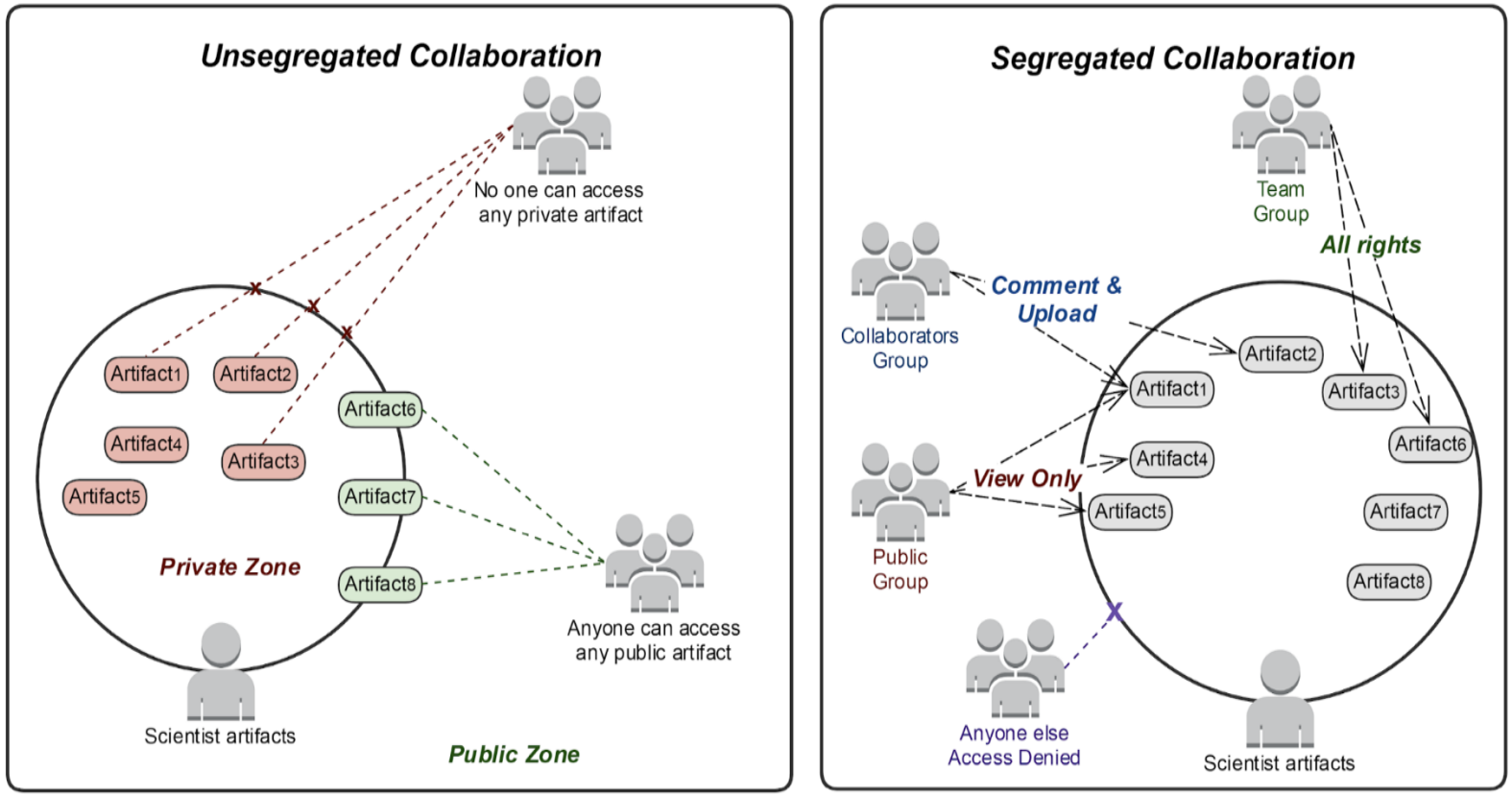

Figure 3.5 Unsegregated and segregated Collaboration scopes: In unsegregated mode, scientists can either grant access to anyone or refuse access to all. With a segregated mode instead, scientists can allow others to collaborate on artifacts by granting access to specific groups based on the level of actions desired.

3.3.3 Workflow Models (W)

To accommodate the growth in complexity of scientific experimental pipelines, more collaborative Web platforms for reproducible research provide a way to design Workflows from within. Thus, scientists craft their experimental pipelines in their browsers and provide the data and the executable based on their platform's artifact construction approach: in-platform or off-platform as presented previously. Moreover, in the case of in-platform construction of artifacts, most platforms host their own marketplace of modules that scientists can use as the building blocks of their workflows. As such, scientist will have artifacts composed of pre-bundled tasks that are managed by the platforms. Additionally, some platforms, will allow scientists to create custom modules that can be contributed to the marketplaces. Then, others can use these custom modules in their own workflows. Alternatively, in platforms offering off-platform construction of artifacts, the workflow definition and its components can be partially created outside of the platform and uploaded at different moments.

One of the most controversially important features of workflow systems is that every module is treated as a black box. As such, only descriptions of the inputs and outputs are provided to link them together to generate the pipeline. Hence, module nodes within research pipelines have a certain level of agnosticism to be able to leverage interoperability. Moreover, while every task runs within its own context (black box), scientists can use any underlying reproducible research software and service to record a reproducible artifact. Thus, another scientist that is more familiar with another reproducible research software can still link it to the latter tasks. Even though we haven't found any use case involving such a combination, we are confident that this feature of workflow systems can be a potential solution to $\mathbf{P_2}$. However, there is a catch. The link between two software wrapped in different ways will require a little more work than the basic effort advertised by the platforms for creating a task. In fact, within the workflow, reproducible research tools inside the task's modules will have to implement a comprehensive understanding of their direct predecessors. They will specifically need to provide enhanced processing capabilities for their inputs which are outputs from other modules.

Based on our overview in the previous paragraph, it is easy to deduce that despite our explanation of how workflows could be used to solve **P_2**, we haven't found any literature reference for such as solution. Furthermore, such a solution does still present a challenge to be implemented. In the following we present eight of the most used reproducible research-oriented web platforms.

- **Citrine.** Developed by Citrine Informatics, it's a web platform to provide machine learning and advanced analytics to material scientists. As such this platform pushes forward for collaboration and the enrichment of dataset manipulations history through materials characterization management, task tracking and data capture. Therefore, this platform is well versed in Web Transformation and Collaboration Schemes. This platform is well known among federal research institutions in the U.S. [**White 2015**].
- **Ergatis.** It's a web-based workflow platform that allows its users to manage reusable computational analysis pipelines [**Orvis et al. 2010**]. Ergatis has in own marketplace of composable modules in the domain of bioinformatics. Scientists can leverage these on a graphical interface to craft their own pipelines. Biological data can be loaded and annotated using the well-established community based schema specification: **CHADO schema** (**http://gmod.org/wiki/Chado_-_Getting_Started** accessed July 17, 2018). Ergatis workflow engine is based on an XML processing core and can be plugged on a compute grid. Thus, it provides enhanced execution metadata and error recovery procedures.
- **Galaxy.** Built by the center for Comparative Genomics and Bioinformatics, Galaxy [Afgan et al., 2016] is a collaborative web platform that allows scientists to perform reproducible biomedical analyses. It is accessible in two ways. The first way is a public centralized web instance (usegalaxy.org) released since 2007. It is a public site that provides considerable CPU power and disk space to thousands of users. The second way is an open source application that can be deployed on any Unix system. Any scientist, team and institution can customize and run their own Galaxy instance. Among the core features of Galaxy are the capability to keep history of the analysis, allow users to manage workflows through a graphical user interface, share and publish their work within Galaxy.

- **HyperThought.** Developed by the U.S Air Force Research Laboratory (AFRL) researchers, HyperThought is a framework with a web aspect that aims at improving the recording, sharing and access of research data. The web aspect of this framework is called ICE for Integrated Collaborative Environment. It has been tremendously used internally at AFRL by scientists specially among Materials and Manufacturing domains.
- **MATIN.** It's an e-collaboration web platform that allows scientists to manage databases or repositories, datasets, software and teaching material. This web framework is a giant factory that binds tools from simple executable to entire computational environment as virtual machines in a collaborative way for scientists. Therefore, scientists can run their executable directly or work within a VM or with a Jupyter Notebook. MATIN is a product of Georgia Tech that is accessible through a centralized instance at https://matin.gatech.edu.
- **OSF.** The Open Science Framework [**Foster and Deardorff 2017**] is a web platform aiming at simplifying scientists' collaboration. It provides a cloud-based features to manage research projects in a structured fashion. Moreover, scientists can manage the access to their projects and activate third party integration. Currently OSF supports the integration of Dropbox, Github, Amazon Web Services, box, Google Drive, figshare, The Dataverse Project and MENDELEY. OSF is free, open source and accessible through a centralized instance at https://osf.io.
- **Taverna.** It's an open source workflow management system that allows scientists to design an execute scientific workflows. Taverna [**Oinn 2004**] leverages other tools for workflow management, activity and service management, user interfaces, workflow components and provenance management, web portals and finally computing infrastructure integration. With Taverna, scientists can construct complex analysis on their data within various types of computational resources. Taverna is open source and can be deployed by anyone respecting the Apache License, version 2.0.

The following Table 3.4 provides a list of the most used reproducible research-oriented web platforms and the concepts they implement.

Table 3.4 Reproducible research web platforms

Platform	Web Transformation	Collaboration Schemes	Workflow Models
Citrine	✓	✓	×
Ergatis	✓	×	✓
Galaxy	✓	✓	✓
HyperThought	✓	✓	×
MATIN	✓	✓	×
OSF	✓	✓	×
Taverna	✓	✓	✓

3.4 CONCLUSION

In this chapter we reviewed the current contributions into the corroboration/reproducibility of research results in two parts. We first overviewed reproducible research focus software methods that deal with the core aspect of reproducibility itself. Second, we overviewed reproducible research platform in the lights of the current concepts that may be potential or partial solution to the problems of interest in this thesis.

In Table 3.1, we list a non-exhaustive list of the current software implementing the six methods (Executable Papers, Notebooks, Workflows, Pre-execution Wrapping, In-execution Wrapping and Post-Execution Wrapping). These methods are clustered in the two major categories described (Literature Programming and Execution Wrapping) in the case of general-purpose reproducible research.

In the discussion section of the first overview, we address the limitations of the current reproducible research focused tools with regard to the problems of interests in this thesis. While

numerical assessment addresses **P_4** partially, other general-purpose reproducible research methods barely address **P_1**, **P_2** and **P_3**.

Finally, we propose to send the reader back to the introduction of **P_1**, **P_2**, **P_3** and **P_4** in chapter 2. While some of the existing software and platforms nor their underlying methods and concepts partially address some of these problems, none of them uniformly addresses them.

While **P_1** might be currently partially addressed in isolation to the other problems, the ultimate goal of this thesis to deliver a unified solution that also covers **P_2**, **P_3** and **P_4** which is not addressed neither here nor in the following concepts as we will present. In fact, as far as the literature is concerned, interoperability in both reproducible research software and platforms is still an open problem that we address in this thesis. Moreover, we have developed on the fact that an effective solution to some of the problems exposed must also solve others in order to be viable. This brings us to state that a solution to **P_3** must also solve **P_2** to be a complete solution to **P_3**.

The following chapters present the two major contributions of this thesis. We enroll the reader in the narrative of the mission we have set to deliver a unifying solution to the exposed problems in two complementary investigations.

Chapter 4 – CORR, THE CLOUD OF REPRODUCIBLE RECORDS

4.1 INTRODUCTION

Plainly showed in the previous chapter, the current solutions to reproducibility can be summarized in terms of the motivations fueling their creators at the time of their inceptions. In fact, the reality of these tools and platforms existence is that their Research and Development (R&D) is guided by the need of the team and the agenda of the Institution, University or Laboratory. As such, general-purpose tools mostly tend to focus in getting closer to the internals of reproducibility issues and resolve them in a way that suits most their current use cases. The platform-based solutions on the other end scale a few steps further with the goal to accommodate more scientists in a more collaborative fashion.

Consequently, most general-purpose tools do not address $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ and $\mathbf{P_4}$. Moreover, while platform-based solutions partially address $\mathbf{P_1}$ and $\mathbf{P_3}$, they fail at covering $\mathbf{P_2}$ and $\mathbf{P_4}$. Hence, in the current literature we have $\mathbf{P_1}$ and $\mathbf{P_3}$ that remain partially unsolved while $\mathbf{P_2}$ and $\mathbf{P_4}$ are left open.

In the present chapter, we present CoRR as a solution to $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$. The unique nature of this solution is that it solves these three problems completely. Moreover, it provides a unifying way to do so. In the process of developing such a solution we must guide the reader through a new terminology understanding beyond the current confusion. This is our attempt to harmonize the various concepts in reproducible research. Then, in the rest of the chapter we present the features that makes CoRR an effective solution to three of the problems aimed for in this thesis.

4.2 ON THE TERMINOLOGY CHIMERA

Lengthily evocated in the problematic statement, there are currently nuances in the notions of corroboration that is impeding our understanding of each other. The three major nuances beneath the concept of corroborating one research investigation host more than one form. As explained in chapter 2, these terms were borrowed and recycled by different scientists, teams and communities. Thus, the first form relates to the borrowed fact of words with a pre-existing meaning not quite unilaterally adopted. The second form relates to the fact that in the literature, we find contradictory meanings. Indeed, some publications clearly redefine one term to the

meaning of another by other publications [Barba 2018]. Based on the Merriam Webster International Dictionary cited in chapter 2, the first form taken by the three terms in correspondence to corroboration have overlapping meanings. Figure 4.1 sketches the mingled interpretation.

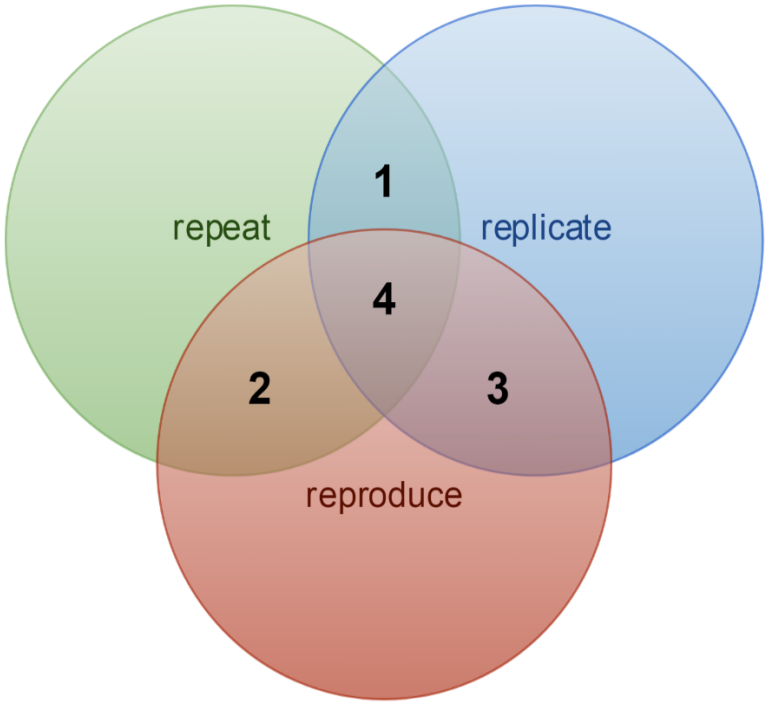

Figure 4.1 Overlap meanings of the three terms: From their original meaning in the English dictionary, there is a subtle overlapping meaning to the three terms used when referring to corroboration.

In **1**, repeat can be confused with replicate [**Bell 2016**] when being thought as stated again or as a copy. Therefore, in that sense a repeated result means a replicated result. In **2**, repeat can mean reproduce [**Bartlett and Frost 2008**] when thinking in term of respectively stating again or causing again. Thus, a reproduced result is a repeated result. In **3**, replicating a result can be understood as reproducing a result [**Duvendack et al. 2017**] if though as copying or duplicating for one and causing again or anew. In 4, the three terms are confused for one another. In fact, looking at **1**, **2** and **3**, one can easily point to the fact that when it comes to meaning corroboration at the strictest sense (copy, state again, cause again), there is a great deal of confusion from one article

to the other. In 4, we place the current global confusion behind the general notion of corroboration. In fact, it encapsulates, based on the needs, the capability to repeat, replicate or reproduce scientific results.

Instead of expanding the current terminology as most scientists will tend to do [**JCGM 2008**], we ought to look for the counter-intuitive. In fact, we seek compressing it to a common sense understanding that it can be expressed in different words depending on the field and the use case.

The three base terms, independently from their extensions, are used to express various meaning in the aspect of corroborating scientific results. Among these terms and based on their original definition, reproducibility is the most accommodating of the three. In some sense it almost provides the sentiment of equivalence to the notion of corroborating in a general sense. Thus, when referring to this term in this manuscript, we actually mean corroboration in a more modern way of caricaturing it. However, we propose to the reader to consider the following ambiguity avoidance from what is currently happening.

- **Strict Corroboration.** We refer to this when everything that concurred to the corroborated item is preserved as is. In the case of a scientific result, as soon as a computation was done, we must have all ingredients to do it again and obtain the same result. This is what most papers aim for when referring to repeat or in the original sense as stating from memory.
- **Loose Corroboration.** We recall to this aspect when we can afford to tolerate a few variations, such as outsourcing the execution to another team with a possible change in material, operating system, etc. This is the meaning mostly attributed to reproducibility as a term.
- **Open Corroboration.** The meaning to this is to indicate when the goal has been to double back a finding by using a different input or fundamentally switching to a new process and method. Replication in its original meaning as creating a facsimile or a similar outcome fits well in this notion.

These three notions are purposely defined to be related to corroboration. Indeed, when detached from it, as done with the current raw recall to repeat, reproduce and replicate it is very much ambiguous to understand the sense of what is meant in general. When we say to have "repeated" a result, we in fact mean that in the process of corroborating a result we were able to strictly recreate it. Thus, repeatability is the process of strictly corroborating a result. Reproducibility is on the other end the process of loosely corroborating a result. And finally, replicability is the process of openly corroborating a result by following another path that fundamentally differ from the original one.

Understanding and reasoning about reproducibility is not as trivial as one thinks. To achieve our unification goal, we must educate ourselves on commonly acceptable ways to express the general notion of corroboration and its subsequent meanings. Therefore, we think the present section is an important aspect of this result. Thus, we expect it to scope more precisely the reader thoughts on the core terminology agreements used during this thesis.

Once settled down on the terminology, we can move toward the most important aspect of corroborating a computation. What is a computation? Under the common notion of execution context, most tools presented in the general-purpose section in chapter 3 have a different answer to this question. To succeed in solving $\mathbf{P_2}$, CoRR must address this aspect in a novel way that will encompass all existing understanding. The next section elaborates lengthily into how we thought best do this: a morphing execution context representation.

4.3 THE EXECUTION CONTEXT IN CORR

4.3.1 Introduction

In the process of corroborating a computational result independently of the level of such corroboration (strict, loose, open), one cannot escape to answer the question of how to represent the execution context that produced such result. In fact, when a problem occurs, one must dig into the execution context to investigate:

- What caused it to change: if trying to repeat.
- What concessions cannot be made: if trying to reproduce.

- What changes are creating too much divergence: if trying to replicate.

Within the current literature, as presented in chapter 3, most methods that focus in helping scientists directly corroborate results had to deal with this representational requirement. Thus, we have enumerated three types of representations currently in used within most scientific tools, services and platforms. They are: metadata focused, raw data focused, and hybrid data focused.

- **Metadata.** This type of representation tends to focus in capturing information in a literature programming fashion. Thus, the goal is that this information could be understood by another scientist or tool and used to construct the corresponding context.
- **Raw data.** This representation holds the copy of all computer related elements involved in the computation that can be copied. As such, the way it has been copied and stored is proper to each tool and platform as each has its own way of recreating the context mostly automatically. Therefore, it follows a very much non-literature programming philosophy.
- **Hybrid data.** More complex than its two pairs, this representation judiciously couples the latter to deliver the best of the two worlds. As such, they are human readable and at the same time can be automatically processed by tools.

While each of these three present advantages and drawbacks (Table 4.1), the specifics regarding what information is crucial and how it must be stored is as important.

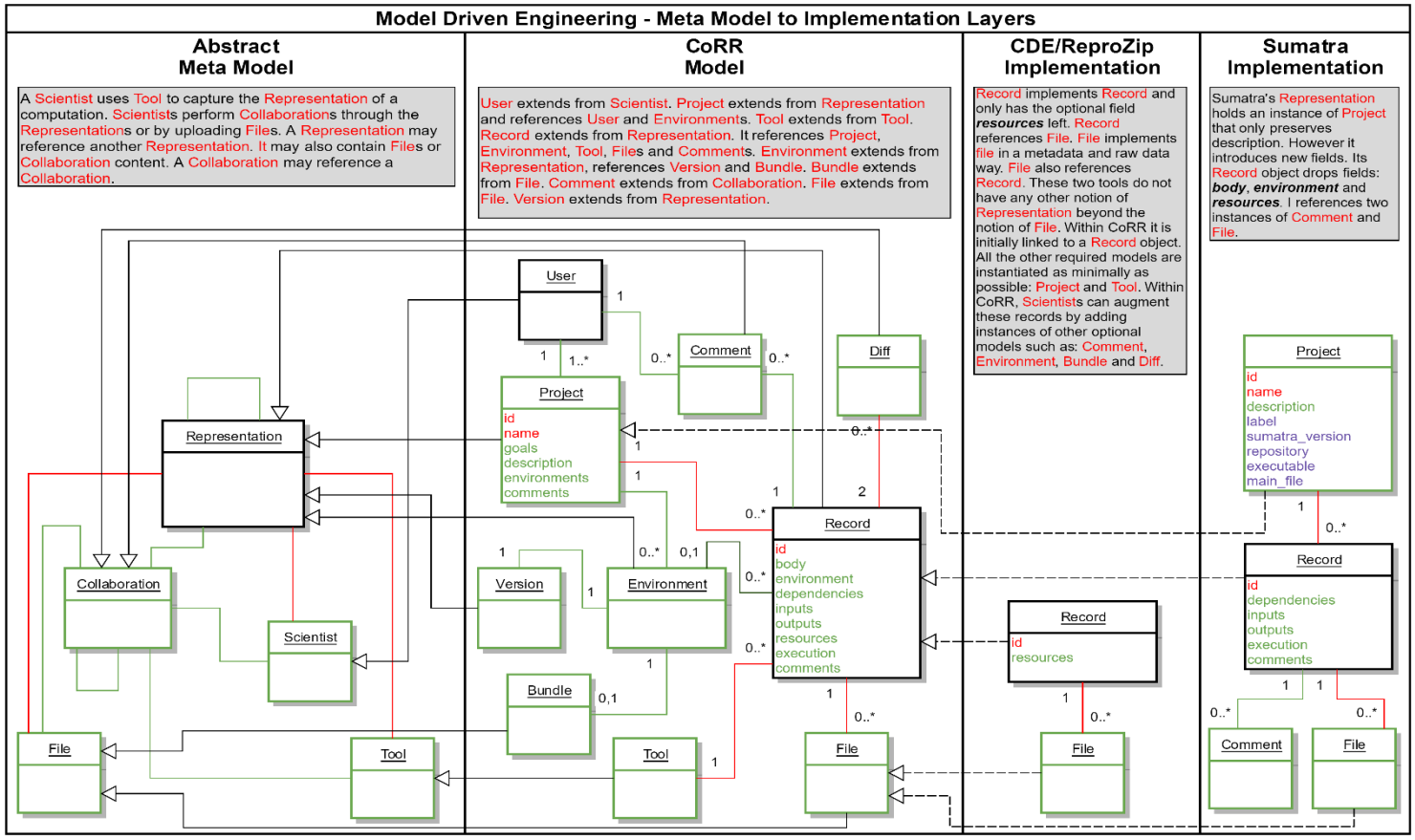

Figure 4.2 MDE reasoning applied to CoRR: It contains the three layer-based transition from which we descend from the meta-model to some implementations through our construct of the CoRR models. Any item in green is optional and is not required. However, attributes and relationships in red are required items. As an example, a File may not exist at all in a representation. However, if it exists it must be referenced by a representation in mandatory fashion or a collaboration in an optional fashion. The purple color is for new items that can be accommodated.

As such we have come to a set of four aspects that must be respected in a representation that intend to unify all others independently of what and how each tool may leverage the three representations types.

Table 4.1 Features values for the three types of context representation modes

Features	**Metadata**	**Raw data**	**Hybrid data**
Human-Readable	Yes	No	Yes
Small Memory Print	Yes	No	No
Self-sufficient	No	Yes	Yes

In our quest of characterizing an ideal representation that is capable of coping with the three existing ones we have leveraged Model Driven Engineering (MDE). Its model-based reasoning has allowed us to produce an abstraction from which we are able to descend to specific case with CoRR aiming to solve P_1, P_2 and P_3. Moreover, we show in Figure 4.2 how this latter case scopes the representation of three existing tools. At first, the abstraction or meta-model specifies the relationship between the actors of the computation execution: **Scientist**, **Tool**, **Representation**, **Collaboration** and **File**. These five models can be extended to take any form the solution requires. For example, within CoRR, Project, Record, Environment and Version extend from Representation while Bundle and File extend both from the more abstract File model in the meta-model.

4.3.2 Open aspect

As shown during this thesis and recalled, reproducibility is a large and often confusing concept. As such, besides systemically capturing everything to guarantee reproducibility, one must know what is crucial and what is not. Indeed, although tempting, capturing everything is not a sustainable endeavor. In fact, a 5 Gigabyte virtual machine image that is being snapshotted every minute will require 2.4192 Petabyte of storage in one year. As a reference of how enormous this is, scientists have estimated the entire world storage capacity from all digital and analogic devices to 295 Exabytes [Hilbert and Lopez 2011]. This is roughly 295000 Petabytes. Thus, the world cannot sustain more than 121942 scientists doing this for more than a year! And this is totally

hypothetical. No one has access to that amount of storage capacity. The second alternative is indeed to choose what we consider as important and enough to represent a computation context. This is what each tool and web platform does and the reason behind the lack of interoperability between tools (**P_2**). Thus, the only alternative to unification is to have a representation that can evolve, it must be open to the point of being easy to extend and readjust. In contrast, having a rigid representation even if it helps enforce some rigor and rush into a formal standard, will probably break sooner than later. An open and flexible representation model is the most sustainable approach.

4.3.3 Corroborative aspect

A software execution context representation that is able to encapsulate all representations from existing tools and platforms must deliver all the spectrum of corroboration aspects. In fact, it must allow scientists to repeat, reproduce and prove that what they are doing is a replicate of the former. This is possible initially by allowing the flexible representation to be transformed into any of the existing representations. For example, a CoRR representation of a simulation pushed by ReproZip can be downloaded and used with ReproZip. As such, CoRR augments the initial representation. It will preserve the existing corroboration features from the tool and if possible add missing features. A unifying solution cannot provide a representation that is not practically used back by at least the originally submitting tools or platforms. We invite the reader to imagine for a moment a case in which a representation of an executable paper become non-executable anymore. Thus, such a representation despite its added features loses one of the most fundamental features which is backward compatibility.

4.3.4 Versioning aspect

As introduced in the Open aspect, an execution context that cannot be versioned will most likely not live for long. The reason is simple. It is the core nature of computing itself. The scientists of our time run simulations all the time. First, because the process of obtaining the appropriate code takes many iterations. Second, when obtained, the same code might be run with different datasets or new libraries. As such, computation is very dynamic. Tracking each is additionally useless if we are not able to compare them meaningfully. As such, comprehensive versioning is

critical. Beyond the usual backup, knowing that a computation result has changed because of a change of operating system, library version or dataset is very useful in the scientific pipeline. Here again, we call to the experience and imagination of the reader. Among the dozens, hundreds, thousands or tens of thousands if not millions of simulations launched, how can one remember why they did in one computation compared to another and what was the actual reason of the difference in results beyond the scientific method change? Most of us will agree that this is a hard question to answer and it is hard for any kind of versioning. Source code versioning falls into the same issue. Without the fact that we can know and tell why we are deleting, adding or modifying lines of code, the whole process of versioning will totally lose its sense beyond the default backup mechanism.

4.3.5 Dissemination aspect

What use is a corroborative execution context if it cannot be easily disseminated? Referencing $\mathbf{P_1}$ and $\mathbf{P_3}$, this aspect has effect on how easy a representation type can be used to reach out to scientists and how easily scientists can collaborate around them. Storage is certainly the main concern. An execution context that exposes directly memory consuming content will tremendously imped user experience. First, scientists must download those content, this may relatively take time and bandwidth. Second, before downloading, scientists must have the adequate storage to hold the representations. As such, an effective representation must separate its meta-data content from its actual raw-data which will be the most memory consuming part. Then, the meta-data content should be exposed to scientists and be made available for the scientists in a way that allows an effective decision of its importance before a final decision to download or perform a cloud-based action on the raw-data. Moreover, such a separation, will allow collaboration to occur at the meta-data level and thus cause the least memory footprint.

4.3.6 Summary

The present section overviewed what we think, it constitutes the grail of what is required of a software execution context representation that can be the foundation of a unification scheme. It must be open to accommodate for unexpected changes and evolutions from existing tools and platforms.

Table 4.2 How CoRR features resolve P_1, P_2 and P_3.

P_1	P_2	P_3
CoRR is in essence a Web application and thus benefits from the reach that derives from it. Moreover, scientists have the liberty to make their computation representation records public and share it through a Web link mechanism.	From the unification standpoint and the research for a generic model that can be equivalent to all possible representations, CoRR can act as a bridge between existing tools. Therefore, tools can leverage this feature to interoperate with each other.	In addition to Web sharing, CoRR enables collaboration that are reproducibility assessments of two records. Such an interaction ideally yields a conclusion that two records are repeats, reproductions or replications. Such a result is invaluable.
Web-based Platform and Web sharing feature	**Generic Intermediary Representation**	**Reproducibility Assessment feature**

It must maintain, add or augment the corroborative nature of the captured information: execution. It must be structured in a way to ease its versioning and dissemination. By covering those aspect, we effectively and fundamentally address **P_1**, **P_2** and **P_3** shown in Table 4.2. Or at least, we have a construct that can support solutions/features to leverage one or many of the aspects to achieve such goals. In the following section, we present the result of our investigation in implementing such an execution content representation. Furthermore, we show how this representation is leveraged to effectively solve **P_1**, **P_2** and **P_3** within a Web Platform: CoRR.

4.4 THE ARCHITECTURE OF CORR

4.4.1 An adaptive and open model

Our investigation has proven that the engineering of a representation model that meets our requirements and delivers on the expected features is both a research for design and technology. First, on the design side, it requires the unavoidable determination of the initial form that will take this representation model. Therefore, we enumerate three major groups of objects that

scope the open aspect, corroborative, versioning and dissemination aspects of the execution context we are looking for. These three groups shown in Figure 4.3 are: the corroboration components, the social features and the platform analytics.

The corroboration components contain, first, the objects that put together, yield the open and flexible representation that can support all other three types of representations and even more allow their extension for additional features. The ultimate object that encapsulates an execution context in CoRR is the **record** object. Within CoRR, record objects must belong to a **project** object in a relationship of one to many. The project object serves at identifying the grand purpose behind all the computations. Thus, it is mostly composed of meta-data that stores its name, its goals and its description. The record object is mostly an aggregation from many others. It references the **tool** object that represents the software that created it, the **environment** object within which it was created and multiple **file** objects (input, output, resources) created or used during the execution. The meta-data part of the record object contains labels, the reason of the computation, the status of the computations and other vital information about the computation. The tool object is merely pure meta-data information about the reproducible research support software and the credentials given to scientists to authorize their interactions with CoRR. This vital mechanism will be explained further. The environment object is composed of meta-data describing the system in which the computation was executed. As such, it contains hardware information, operating system information, configuration and libraries details. Also, the record can provide a **bundle** object for the environment. It is an object that contains both specific meta-data and actual raw content of the environment. For example, a bundle of a docker image can be provided as the environment. Thus, the meta-data will be providing details about what a docker image is. Finally, a file object similarly to a bundle object is designed to represent any file uploaded to the record as an input, output or resource of the computation.

While these three groups might be useful in our conception of how CoRR should be specially in terms of collaboratives features, only a few models are critical to achieving our investigation. They are shown in Figure 4.2 which displays how they related to our abstraction. In fact, while Figure 4.2 shows CoRR models in the bigger picture of how they came to be, Figure 4.3 narrows

it to how it has been applied. The models that are not listed in 4.2 are: Message, Profile, Stats, Traffic and Access.

Although part of the notion of collaboration, Message has been left out of Figure 4.2 for simplicity due to its versatile usage possibility. In fact, scientists can communicate through messages on subjects irrelevant to any project or record.

Similar to Message, Profile has been taken out from Figure 4.3 as it is not a critical model to extend the Scientist meta-model from.

The models Stats, Traffic and Access are purely administrative concepts are not even conceptualized in the meta-model abstraction.

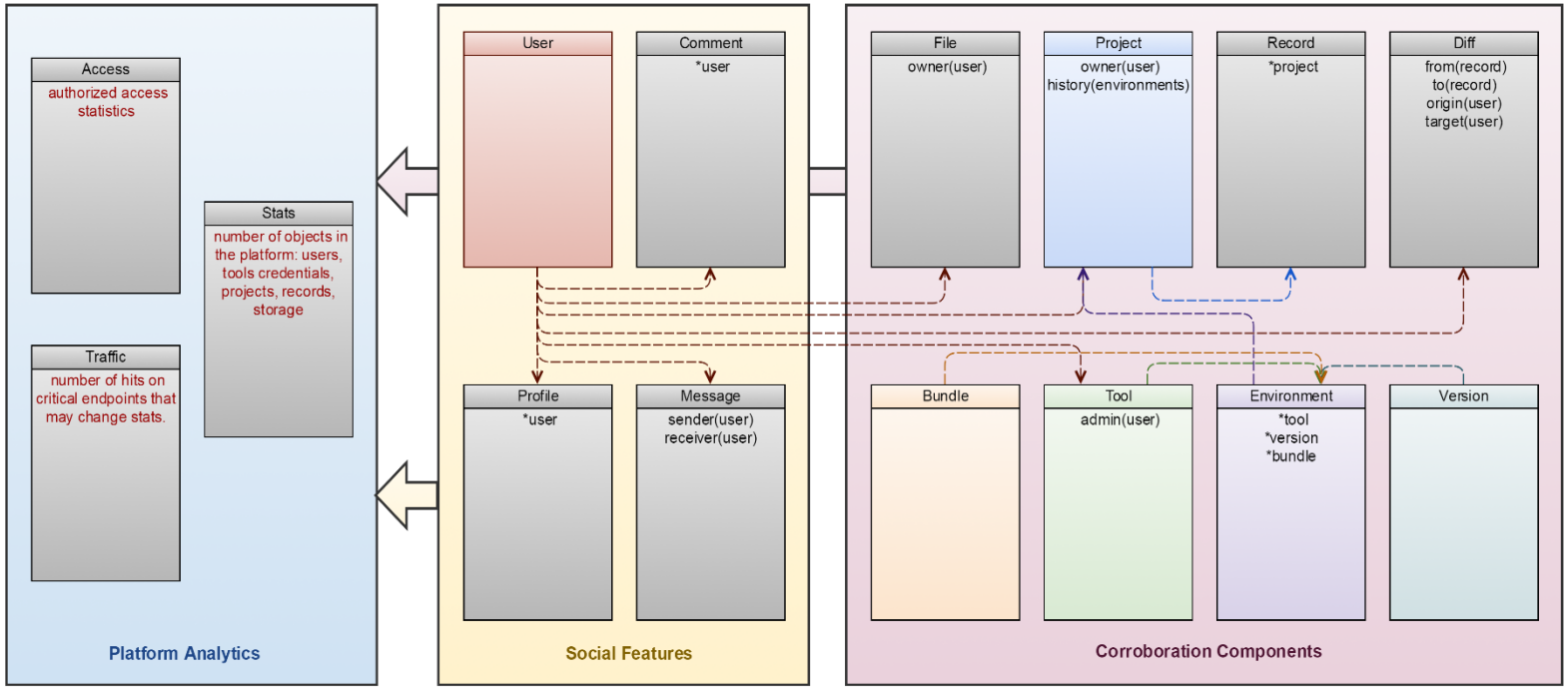

Figure 4.3 CoRR models of three groups of components: The platform analytics components allow the storage of three different statistics on the platforms. The platform social features focus on components that handle user information. The third group contains the core components of the platform.

They are useful in practice to the manager of the CoRR instance only and does not add nor reduces in any way scientists experience. Moreover, it does not have any added value to the features solving P_1, P_2 and P_3. However, in modern design of cloud platforms, these analytics features used for the measurement of the solution impact to society in terms of adoption and contribution are fundamental. The other two groups concurring in the representation of the execution context in CoRR contains objects that construct the collaborative nature embedded into scientific inquiries in the platform. The social features category concerns objects that focus on user's information and communication within the platform. The platform analytics category contains the models that are dedicated to capture filtered and useful analytics on interactions between the users, the tools and the platform. While the platform analytics focuses in quantifying this activity, the social features focus in the scientists' identifications and their core interactions. Thus, the record object withdraws information such as ownership, interactions and statistics from these two groups. The later are indeed part of the foundation for the extended features added by CoRR.

Second, on the technology side, we have faced the question of the adaptive and open nature of our representation. To implement the previously described abstract representation, a question of the technology to use as a database to effectively store these objects had to be answered. We have not dug into the specifics of what should be in each object across the four groups as nothing is mandatory beyond their fundamental relationships. In fact, we are in need of a database technology that will technically allow that the creation of instances of each object be void of what each requires in their initial form and even more. This is shown in Figure 4.2 through the possible implementations of the CoRR models with three tools. If for example a tool decides that its representation of its computations must include the speed of the internet, the new field "internet" should be accepted as a custom field and stored in the record object. To provide the necessary flexibility, CoRR uses **MongoDB** (**https://www.mongodb.com** accessed August 29, 2018), which supports a highly flexible data model and manages complex data model migrations. By using MongoDB, the data model can handle three possible representations of the computation: pure meta-data, file only and a mixture of both. Additionally, by enforcing a consistent mapping between the data representation and its associated tool, CoRR can

seamlessly migrate the data model whenever the representation of reproducible research support tool is updated. Despite this flexibility, the CoRR data representation requires a minimal structure from the data model associated with the given tool. In fact, we have come to a minimal set of requirements and flow of interactions expected from the tools. By agreeing to the following non-intrusive requirements, the tools are expected to keep the same level of flexibility in their designs:

- The creation of a project requires at least a unique name.
- The creation of a record requires at least a project id.
- Files can be uploaded to projects and records under specific groups: resources, inputs, outputs, environments and bundles.

In addition, these requirements allow a more thoughtful and productive conversation between the CoRR research team, the tools research teams and the scientific communities involved. We are looking out to build a sustainable ecosystem that does not focus on a single tool but instead allows any number of tools that follows our basic common rules to interact with CoRR and open doors for interoperability between tools and even more for scientists.

4.4.2 State of the art scalable-federated architecture

CoRR is a web platform composed of five key components as shown in Figure 4.4. They are: a web frontend, a cloud service, an API service, a database service and a storage service. The database and the storage are linked to the cloud and API components. The CoRR database (pointed out previously) is dedicated to managing meta-data content only, whilst the CoRR storage service manages the file storage distinct from meta-data. Currently, CoRR supports both AWS S3 and standard file system for file storage. Support is planned for more storage types such as SFTP and modern alternatives such as distributed file systems. Users interact with CoRR through two main entry points: the web frontend and the API service. The frontend is the web view access exposed to the user. The frontend is designed following **Google Material Design guideline** (**https://material.io/design** accessed August 29 2018), via the Material Design CSS framework. Material design provides a familiar and contemporary look and feel to the web

interface. The CoRR frontend is compatible with most commonly used web browsers. The CoRR API is a HTTP RESTful entry point that allows tools to securely write and read to the CoRR platform. The CoRR frontend forms an exclusive one-to-one link with the CoRR cloud component. CoRR's cloud component is the restricted bridge that the frontend must go through to communicate with the database and the storage. Thus, the cloud component is an HTTP RESTful service that is dedicated to the frontend only while the API is dedicated to tools. CoRR components are designed in a standalone fashion as microservices.

By logically linking its components, CoRR benefits from all the advantages of a modern microservices oriented web platform such as scalability, high availability and flexible deployment. Moreover, the judicious coupling of the frontend and the cloud component enhances federated capabilities in which access to an instance could give the user search capabilities across all the other open CoRR instances.

In the following sections, we provide details on how CoRR frontend visually displays its four major elements:

- **Tool.** The element that contains the credentials of a supported and authorized tool. Administrators must create tool object instances to authorize access to tools through which scientists can connect to CoRR.
- **Project.** The element that groups records and allows users to cluster computing campaigns. Projects can be created both from the web or using an authorized tool.
- **Record.** The element that stores a computation in CoRR. Records can be manually built through the web or automatically created using a tool.

❖ **Diff.** The element that holds a reproducibility assessment which is a one-to-one relationship between records (see the section on “CASE STUDY” for more details).

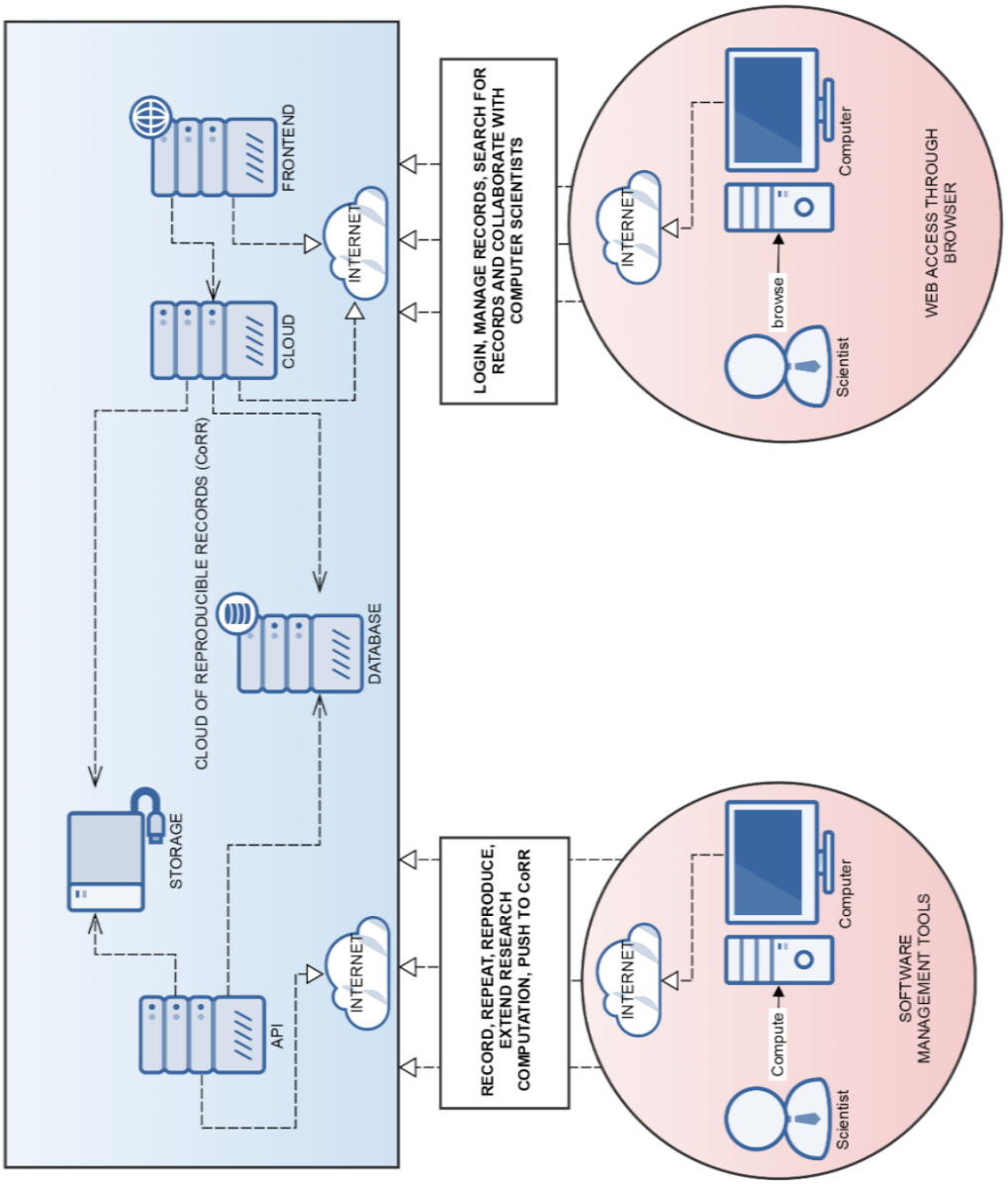

Figure 4.4 CoRR’s platform architecture: It shows how the five components (API, STORAGE, DATABASE, CLOUD, FRONTEND) are connected. It also shows the two entry points (API, FRONTEND) and the actions available to the scientists. depending on which way they are accessing CoRR.

4.4.3 The main elements of views in CoRR

The CoRR frontend uses a common visual structure to display its four main elements. This common structure is composed of three groups of interactives action items. From bottom to top, the first display group is the references area. It shows statistics or references to other elements related to the one being viewed. For example, the record element is related to some input’s files,

outputs files and some dependencies. Hence, the record element's bottom view displays the number of inputs, outputs and dependencies. The second display group is the content area. It only shows the most important fields of the element. The content also varies from a major element to another. Thus, while the tool content displays its creation date, name, status, key and description; the project content displays its name, creation date, access, tags, goals and description. The last display group is the toolbox area. It provides actions to be performed on the element and its content. Such action items are:

- **delete.** an action that allows the element's owner to delete it. All elements have this action item.
- **update.** an action that synchronizes the local changes on the element's fields with the database. All elements have this action item.
- **share.** an action that produces a sharable single page link of the element. All elements have this action item.
- **download.** an action that produces a file that represents the element. The project element is the only element that does not currently have this action item.
- **upload.** an action that allows the element's owner to add content to the element. The project and record elements are the only one to provide this action item.
- **environment.** an action item that allows the construction of an environment. The project element is the only one to allow such an action item.
- **user.** an action item that displays the element's owner short bio. This action is typically provided on the diff element. Yet, when logged in as administrator, this action is provided on all elements.
- **project.** an action item that displays the related project. This action is only provided by the record element.
- **select.** an action item that allows the selection of an element. It is only allowed on the record element to build "diff" elements.
- **key.** an action item that allows only the administrator to renew the tool element key.

In the following parts, we invite the reader into a more detailed acquaintance of these four crucial concepts of CoRR visual innerworkings.

Concept of tool

Derived from the tool model, the tool view is a component that holds information about tools supported on a CoRR instance. As shown in Figure 4.5, this view contains three types of view elements.

The top section is the toolbox area. It contains clickable icons. It allows the user to perform actions on the credential object. In this case from left to right, the user can firstly share the credentials and secondly download the credential. The share icon produces a public URL that can be easily distributed. This URL leads back to a single page rendering the credential view. The download icon results in the production of a JSON file that all supported tools must have to configure themselves to communicate with a CoRR instance. The middle section contains information about the tool (name and description), the tool credential key and its status. The tool credential key is a SHA-256 hash. The status of the credential is activated when usable and deactivated when unusable. The bottom section holds the statistics about the credential usage. From the left to the right, we first have the number of scientists that use the credential. Then, it shows the number of projects created with this credential. And finally, it displays the number of recorded computations done with the credential. The tool credential top section allows more actions to the administrator. For instance, they have three extra icons. One for editing the content of the middle section, a the second one for renewing the credential key and a last one for deleting the credential.

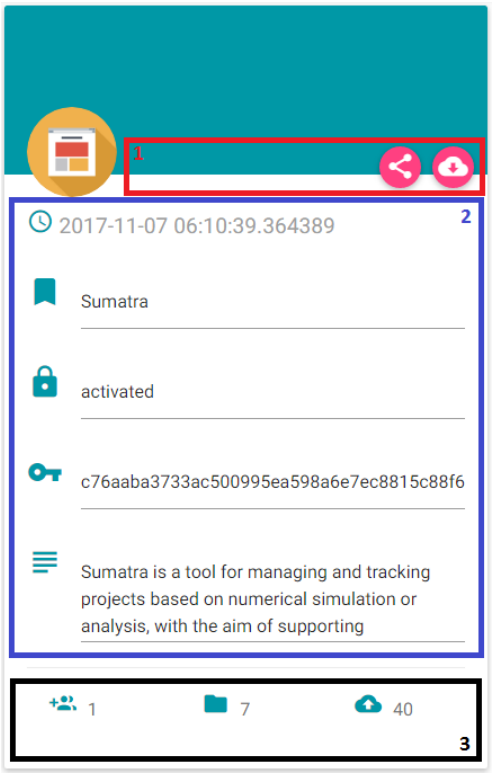

Figure 4.5 Visual representation of a tool object in CoRR frontend

Concept of project

Before being able to store computations in CoRR, as explained when mentioning the minimal requirements in the previous sections, a project must be created. For the computational scientist, a project is the main piece of code/executable on which the recording tool is expected to focus on. The project view also contains three types of view elements as shown in Figure 4.6.

The top section contains the toolbox for altering the project content from the web. From left to right it firstly allows the user to get a shareable URL for dissemination beyond CoRR. The second action allows the update of some of the content in the middle section. The third action let the user provide meta-data and data about a computational environment to be uploaded. The project owner can for example, provide description only information about the operating system and the hardware used as a document or upload virtual machine/container images. The fourth action gives the user the capability to create a record from the web. Finally, the last action, allows the user to delete a project and all its content. The middle section contains two types of

information. The project name and the date of creation cannot be modified from the web. On the other end, the project privacy state, the labels, the description and the reason can be edited by the project owner when clicking the edit action in the toolbox section. The bottom section contains three statistics about the project. From left to right, it firstly shows the amount of computations recorded in this project. Then it displays the number of reproducibility assessment done on records in the project. And finally, it presents the number of environments uploaded in the project.

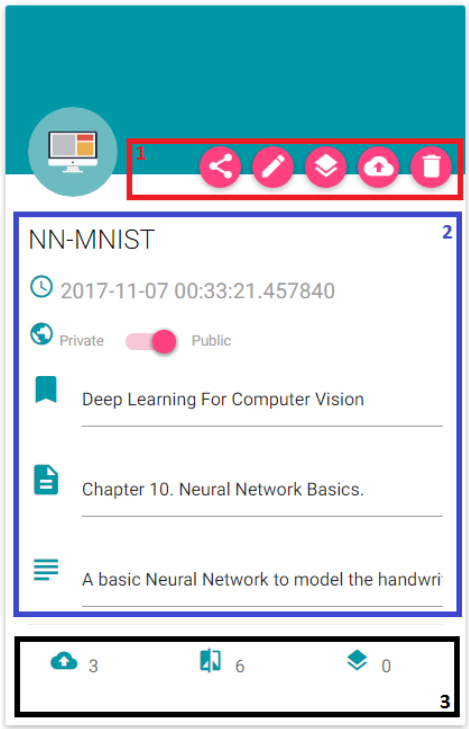

Figure 4.6 Visual representation of a project object in CoRR frontend

Concept of record

A record is the capture of a computation pushed to CoRR. Records are stored in CoRR within their associated projects. The record view is also composed of three sections as shown in Figure 4.7.

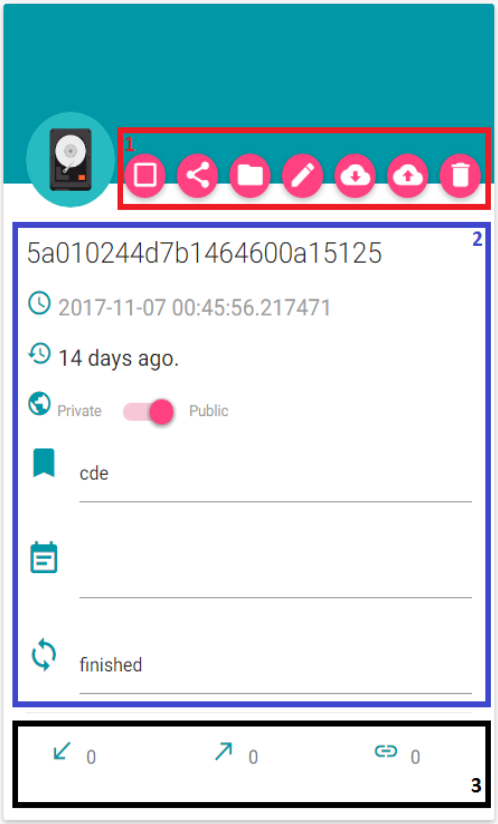

Figure 4.7 Visual representation of a record object in CoRR frontend

The top section, from left to right allows the record owner to select the record. The selection of two records is required to create a reproducibility assessment. Then, the second action item generates a shareable URL of the record. The third action allows the user to view the project from which this record depends on. The fourth action allows the owner to edit parts of the content in the middle section. The fifth action allows the user to download the record. The download of a record from CoRR results in a compressed file that contains a self-descriptive content with metadata in JSON and actual data files.

The metadata JSON files are:

- **application.** metadata about the tool used to record the computation.
- **project.** information about the project to which the record belongs to.
- **environment.** uploaded environment metadata associated with the record.
- **body.** important fields that define a record in CoRR.

- **record.** extra metadata fields that are supplementary to the body. It includes fields that are tool specific.
- **parent.** id of a record that is the parent or the predecessor of the current one. This helps support complex workflows records.
- **execution.** metadata of the command used to execute the computation.
- **dependencies.** metadata of the libraries or dependencies required by the computation to execute.
- **inputs.** metadata of the input files ingested by the computation.
- **outputs.** metadata of the output files produced by the computation.
- **resources.** any other meta-data of general data files that must be associated to the computation for some reason.
- **comments.** communications in terms of comments sent by users on the record.

The corresponding data files that are associated with the metadata files in inputs, outputs or resources, are named in a CoRR record using the following pattern:

- **input.** input-recordID_fileName.fileExtension
- **output.** output-recordID_fileName.fileExtension
- **resource.** resource-recordID_fileName.fileExtension

We invite the reader to download a record for corroboration on the NIST instance of CoRR at **https://corr.nist.gov**. The sixth action item allows the record's owner to alter the record metadata through JSON, YAML or XML format. The record owner is also able to upload data files such as inputs, outputs and resources. Finally, the seventh action item allows the record's owner to delete it with all its associated content. The middle section of a record shows its id, its creation date and the last update time. This metadata cannot be edited from the web. This section also displays the privacy status, the tags, the rationales and the computational status of the record. This metadata can be altered with the edit button on the top section. The bottom section of a record displays from left to right the number of input files, output files and finally dependencies.

Concept of diff

A diff in CoRR is the component responsible for storing a reproducibility assessment that links a pair of records. These assessments are based on the terminology contextualized here: replicated,

repeated, reproduced and their respective opposites. The diff view like the other concepts views is composed of three sections as shown in Figure 4.8.

The top section contains the set of actions that the record owners can perform on the diff. From left to right, the first action item produces a shareable URL for dissemination purposes. The second action allows one of the diff contributors (referenced records owners) to edit parts of the middle section. The third action is for downloading the diff. It consequently results in the download of the two records involved in the diff. The last action item allows one of the diff contributors to delete it. The middle section shows the diff id in CoRR and the time of its creation. These two cannot be altered from the edit button in the top section. This section also displays the assessment method (default, visual or custom), the assessment scope (repeated, replicated, …) and the status of the assessment (proposed, agreed, denied). These fields can be updated by one of the two contributors. The bottom section displays from left to right, first the record from which the assessment is initiated. Secondly, it shows the record to which the assessment is created and finally the number of comments made on the assessment.

In the case study, we present in section 4.5, we give examples of reproducibility assessments creations and how they are interpreted. The current version of CoRR restricts their creations to the creators of the records only. We plan to open it to all users in a later version.

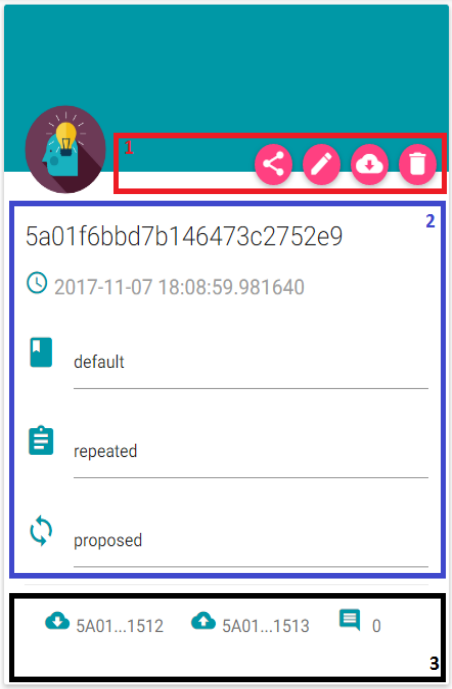

Figure 4.8 Visual representation of a diff object in CoRR frontend

4.4.4 Instance Administration

By briefly recalling the architecture subsection on "State of the art scalable-federated architecture", it derives from it that the CoRR deployment is up to date in terms of DevOps best practices. A CoRR instance can either be managed as a set of Linux services or a set of Docker containers. Automated and configurable deployment schemes reduce greatly the amount of work needed to setup an instance. Moreover, the CoRR instance at **https://corr.nist.gov** is a Docker container based deployment while the one at **https://corr-root.org** is a Linux services based deployment. During the deployment, an administrator (admin) account credential is required to create the first account. This admin has the following roles: user management, content management and tools credentials management. Regarding the user management, the admin is required to approve users accounts upon registrations, manages their access rights and moderate their storage limits. The admin has the rights to delete or alter any content on the instance. To be able to push content to a CoRR instance, a tool is required to hold two 256-SHA access credentials. The first credential is the **user API key**. The second credential is the **tool API**

key. The administrator of the instance is the only one capable of issuing tool credentials and renewing their keys. A tool is denied access to push content into a user's account without this pair of keys.

4.4.5 Main User Features

By registering in a CoRR instance, scientists are given two ways of interactions. The first interaction entry point is the web. From the home page, they can first manage their accounts information and media. Then, from the dashboard page, the scientist can view the authorized tools and manage projects, records, environments and assessments.

The current features described here constitutes CoRR version 0.1 shown in Figure 4.9. A search capability with excluding filters is available not only from the home page, but also from the connected user dashboard. The second interaction entry point is with a CVC tool. The number of features provided by the tool is not constrained by its integration to CoRR. Thus, tool features vary from one to another and follow their own roadmaps. We have come to the resolution that it has enough functionality for initial usability.

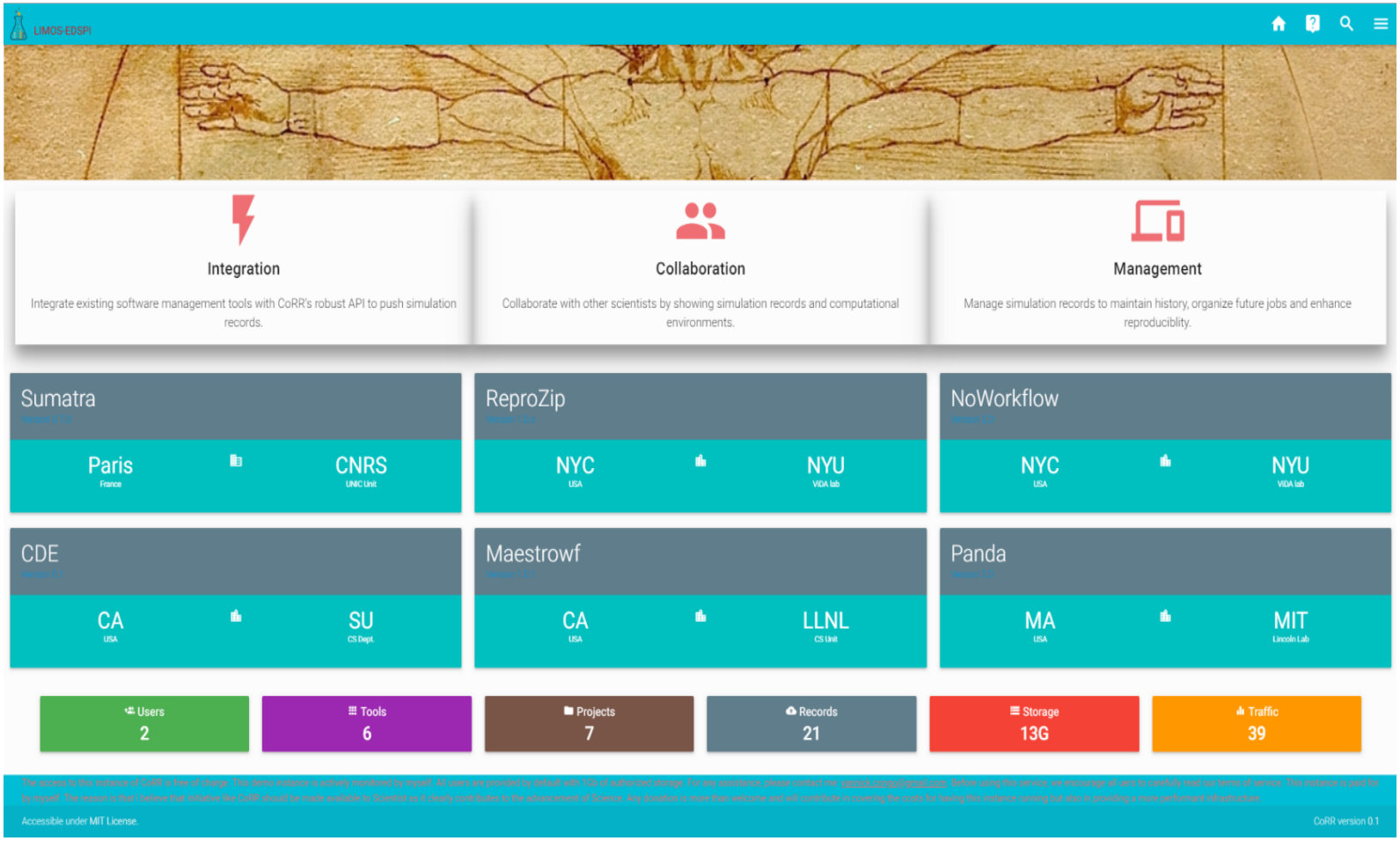

Figure 4.9 CoRR version 0.1 home: The home page of a CoRR instance. It shows the main features of the platform, a list of the supported tools and statistics regarding its usage.

4.5 CASE STUDY

In this case study we compare the integration of the three most used tools (Sumatra, ReproZip and CDE) with CoRR. The comparison is based on the computation of four Machine Learning examples. In the following, we first describe the examples in question. Then, we show how these examples are computed using these three tools. Subsequently, we provide their resulting CoRR records. Third, we present the features used to compare the tools integrations to CoRR. We also provide an aggregated view of the results (see Table 4.1). Finally, we provide a summary of this case study which includes the important notion of reproducibility assessment in CoRR. We note for reproducibility concerns that the computation of all the examples in the machine learning book's Chapter 10 [**Rosebrock 2018**] took 13GB in CoRR and 16 hours on a ThinkPad model P50s. Moreover, we made available a GitHub repository (**https://goo.gl/K2Z4BY** accessed September 26, 2018), that will help the readers install the CoRR integrated version of Sumatra, Reprozip and CDE. The readers will find directions in the repository README. Thus, the readers will be able to register, to access the shared instance of CoRR in this article, and to execute the examples we computed in this case study while storing their records in their personal user spaces within CoRR. Also, the reproducibility assessments and all the other materials produced in this case study can be found by search in CoRR.

4.5.1 Examples

This section presents the two pairs of examples used in this case study. The first pair focuses on Artificial Neural Network (ANN) models learning the logical Xor, while the second focuses on more complicated ones recognizing handwritten digits.

The first pair of examples models the logical exclusive OR of the corresponding bits from two binary strings. Modeling such an operation can be part of the fundamentals of learning ANN techniques. As such, Dr. Rosebrock (the author of the examples' book), provides two codes that model the Xor bitwise operation.

In the second pair of examples, Dr. Rosebrock explains how to model the MNIST which is a collection of handwritten digits. Recognizing digits and in general any characters/drawing is an

active research area that is recently witnessing many contributions from the Machine Learning community.

A Neural Network Model of Xor

Like the previous example, this is also implementing an ANN model of the Xor operation. Yet, differently than the previous one, which rely on a Perceptron approach, this example uses a backpropagation technique with extra layers in a more complicated Neural Network. This example can be viewed as a replicate of the previous one. But one that outperforms its predecessor because it can properly model the Xor operation.

A Neural Network Model of MNIST

In Dr. Rosebrock book, this first technique to model MNIST is based on a custom Neural Network example using a backpropagation technique. This technique allows the model to reach an overall accuracy of 98%. The following subsection presents another example provided by Dr. Rosebrock that uses a different technique.

A Keras Model of MNIST

In this second example aiming at modeling the MNIST dataset, Dr. Rosebrock shows the usage of a Deep Learning library called Keras. By using the Keras library, Dr. Rosebrock implements a technique using a feedforward approach. While the previous example can obtain 98% of accuracy, this example is only capable of reaching up to 92%. Furthermore, the author prepares the reader for a more suited type of ANN capable of reaching an overall accuracy of 99%: CCN (Convolutional Neural Networks). The latter is not discussed in this case study. This example and the previous one can be interpreted as replicated techniques (like the previous pair) with the first one outperforming the second this time.

In the following three subsections, we show how the three most used tools are used to run these examples and record their computations in CoRR.

4.5.2 Computing with Sumatra and CoRR

Before running the examples, we create four directories. Each directory contains the appropriate code of each example. To run the four examples with Sumatra, we achieve the following set of actions: put all the directories under version control, setup Sumatra in each folder, run each code with Sumatra to capture the computation and push it to CoRR. The exact generic commands are provided in Code 4.1.

```
$ git init; git add –all; git commit -m “begin”
$ smt init -s config.json project_name
$ smt config --executable=python --main=code.py
$ smt run
```

Code 4.1 Example of commands to configure Sumatra: The first line shows the version control setup. The following two lines initialize Sumatra by providing the necessary parameters in “config.json” and what to execute. The last line direct Sumatra to launch the computation.

Sumatra and all the following tools require the config.json file when used with CoRR. It contains the credentials they need to access CoRR. Sumatra produces a local hidden folder on the host computer (named smt) which contains a JSON file named project and a SQLite database named records. The project file holds metadata about the project while the database stores metadata about the computations. Sumatra makes copies of the output files only if placed in a folder named Data. Every time that a computation is done another set of rows is added to the tables in the record database. Thus, Sumatra’s local storage is kept and updated when also using CoRR.

4.5.3 Computing with Reprozip and CoRR

Differently from Sumatra, ReproZip does not require any prior setup. The configuration, the execution and the capture are all done in the same command as shown in Code 4.2.

```
$ reprozip trace -config=config.json -name=project_name python code.py
```

Code 4.2 Example of commands to configure ReproZip: This single command both configures the tool and directs it to launch the computation.

ReproZip produces a hidden folder on the host computer (named reprozip-trace) which also contains two files: a YAML file named config and a SQLite database named trace. The YAML file stores all the necessary configuration metadata to construct the computations records. The database records all the additional metadata required to generate the ReproZip computations records as compressed files. For every new computation, ReproZip asks the scientist if the new content should be appended to the previous one or erase it.

With the metadata hosted in the YAML file and the snapshot of literally everything linked to the computation, CoRR represents a ReproZip computation with some metadata but also the entire computational environment involved.

4.5.4 Computing with CDE and CoRR

CDE is very much like ReproZip in the method used to capture the computational environment and the unnecessary need for a prior setup as shown in Code 4.3.

```
$ corr-cde --config config.json --name project_name --cmd= "python code.py"
```

Code 4.3 Example of commands to configure CDE: This single command also both configures the tool and executes the computation.

The YAML file produced by ReproZip allows metadata to be understood by CoRR and pushed in. This specific feature is the major difference between CDE and ReproZip. In fact, CDE captures are only compressed files with no metadata in addition for now.

The main content produced by CDE on the host computer is a folder named cde-package that contains the snapshotted content. Thus, in CoRR, CDE generates records that are purely computational environment snapshots with no metadata as Sumatra and ReproZip.

4.5.5 Results

This case study is provided to the reader to firstly demonstrate how the integration of the most used tools to CoRR does not impact their original behavior. Second, to demonstrate the differences between Sumatra, ReproZip and CDE in terms of local storage and how it relates to the example's records in CoRR. Third, it allows the reader to grasp the challenge of coping with various representations of the same concept. In this case: computations. The result of this

experiment is a set of twelve computations records using Sumatra, ReproZip and CDE on four ANN examples in Dr. Adrian Rosebrock's book. Moreover, we compare these tools based on their integration to CoRR. While we provide an aggregated view of the results of this comparison (see Table 4.3).

The following bullet-points present the features used for this comparison:

- **integration work.** number of lines added to the tool source code to provide its current integration capability.
- **storage used.** storage capacity used by the tool to store the computations records in CoRR.
- **metadata capture.** does the tool representation of the computation contains some metadata?
- **environment snapshot.** does the tool representation stores the snapshot of the computational environment?
- **reproducibility effort.** is the tool representation directly reproducible after a download from CoRR?

Table 4.3 Integration Comparison Results *

Features	Sumatra	ReproZip	CDE
Integration Work	526	234	204
Storage Used	~1GB	~6.5GB	~5.5GB
Metadata Capture	YES	YES	NO
Environment Snapshot	NO	YES	YES
Reproducibility Effort	NO	YES[1]	YES[1]

* Table displays how each tool performs with regards to the compared feature.

[1] Requires a few tweaks to rerun the record.

The aggregated results (see Table 4.1) show the impact of the environment snapshot in the storage being used but also regarding the reproducibility effort. Also, the integration and the metadata capture capability indicate that the latter requires more lines of code than the

environment snapshot. Thus, Sumatra requires more lines of code for mostly metadata capture which leads to a smaller storage footprint. At the opposite of Sumatra, CDE requires less lines of code to focus in environment snapshot, which requires more storage but less effort to effectively reproduce the computation. Finally, ReproZip at the middle of Sumatra and CDE, has the advantage of delivering metadata and environment snapshot for a better insight into the computation and its reproducibility. Yet, its storage footprint is bigger than CDE's.

One of the current challenges in reproducible research is how to interactively and socially frame reproducibility assessments in a digital way. In other words, how can scientist X digitally tell scientist Y that computation result Rx is a replicate, repeat or reproduction of computation Ry?

CoRR is attempting to solve this by providing the concept of a diff (like Git) as a reproducibility assessment proxy. By picking two records, a scientist can contribute to a new layer of assessments. This is a very useful feature to any scientific interests in which an answer to questions such as; "*what are the existing replicates of the Keras MNIST ANN model ?"*; might be relatively important. For this case study, we identify reproducibility assessment contributions that are possible with the twelve computation records.

First, within each of the four projects, all the records (computations) from the three CVC tools are indubitably repeats of one another. This creates three unique reproducibility assessments as "*repeats*" in each project. This leads to another set of twelve reproducibility assessments at the scale of the four projects. Second, since NN-MNIST and Keras-MNIST are two examples of ANNs modeling the MNIST handwritten digits recognition. Hence, their computations are "*replicates*" from one project to the other. In this case, we can make three reproducibility assessments in which a captured computation using respectively CDE, Reprozip and Sumatra in NN-MNIST is a replicate of the other in Keras-MNIST. For convenience, downloading an assessment from CoRR results in a compressed file that contains the two records involved in the assessment. Finally, to come back to the previous question that marks the importance of these assessments, the answer will be NN-MNIST records for now.

The development of CoRR is composed of two major repositories. The first repository (**https://github.com/usinstgov/corr** accessed September 27, 2018) contains the source code of

the platform. The second repository (corr-deploy) contains the mechanisms for customizing the deployment of the platform depending on the infrastructure. In fact, CoRR can be deployed as a set of containers preferably on non-Linux machines or a set of Linux services otherwise. The corr repository is composed of 47 Python files and 14696 lines of code. There are 15 classes that represent the models in the corr-db module shared by the corr-cloud corr-api services. The user interface frontend is composed of 5 main view pages that are dynamically populated by 19 XML files. The corr-deploy repository is composed of 23 configuration files to build a Linux-based services version of a CoRR instance and 35 configuration files for a docker-based containers version. This deployment mechanism takes into account complex requirements. For example, any number of services can be organized at will to deploy on the available servers. If there were 3 servers available, we could do any of the following combinations to fit DB, Cloud, API, Storage and Frontend: 1-1-3, 2-1-2, 4-1-0 or 5-0-0. Moreover, the deployment can be readjusted at any time to accommodate with the load on the whole platform. Practically CoRR addressed $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$ better than any of the existing solutions presented in chapter 3 because it addresses them as a whole as they are inter-dependent of each other for efficiency. For example, CoRR is better at solving $\mathbf{P_1}$ because it addresses $\mathbf{P_3}$ better by allowing advanced collaborative actions through diffs as shown in Figure 4.10 for the use case. Furthermore, it addresses better $\mathbf{P_3}$ because it models are designed to solve $\mathbf{P_2}$. Thus, as shown in Figure 4.2, Tools representations gain equivalence within CoRR. Therefore, collaboration benefits extensively.

It is important to remember that CoRR is a unification-based contribution. Despite the features added to the tools integrated to it, CoRR suffers from what these tools cannot solve at their core. We are referring to the features needed by scientists locally in their computing environments. A major issue is exact numerical repeatability. In fact, as of today, no tool can currently guarantee exact numerical reproducibility from any current execution context representation. The most CoRR can guarantee in that direction is that a representation pushed to CoRR can be reused as if it came straight from the tool itself. CoRR augments the representations without tempering with their inner properties. Therefore, we must stress that a record from CoRR cannot be expected to deliver the same numerical outcome as this is beyond the capabilities of our approach. Such a

feature must be implemented in the tool itself and will most likely change the way the tool approach in representation construction. Thus, CoRR is limited in that sense.

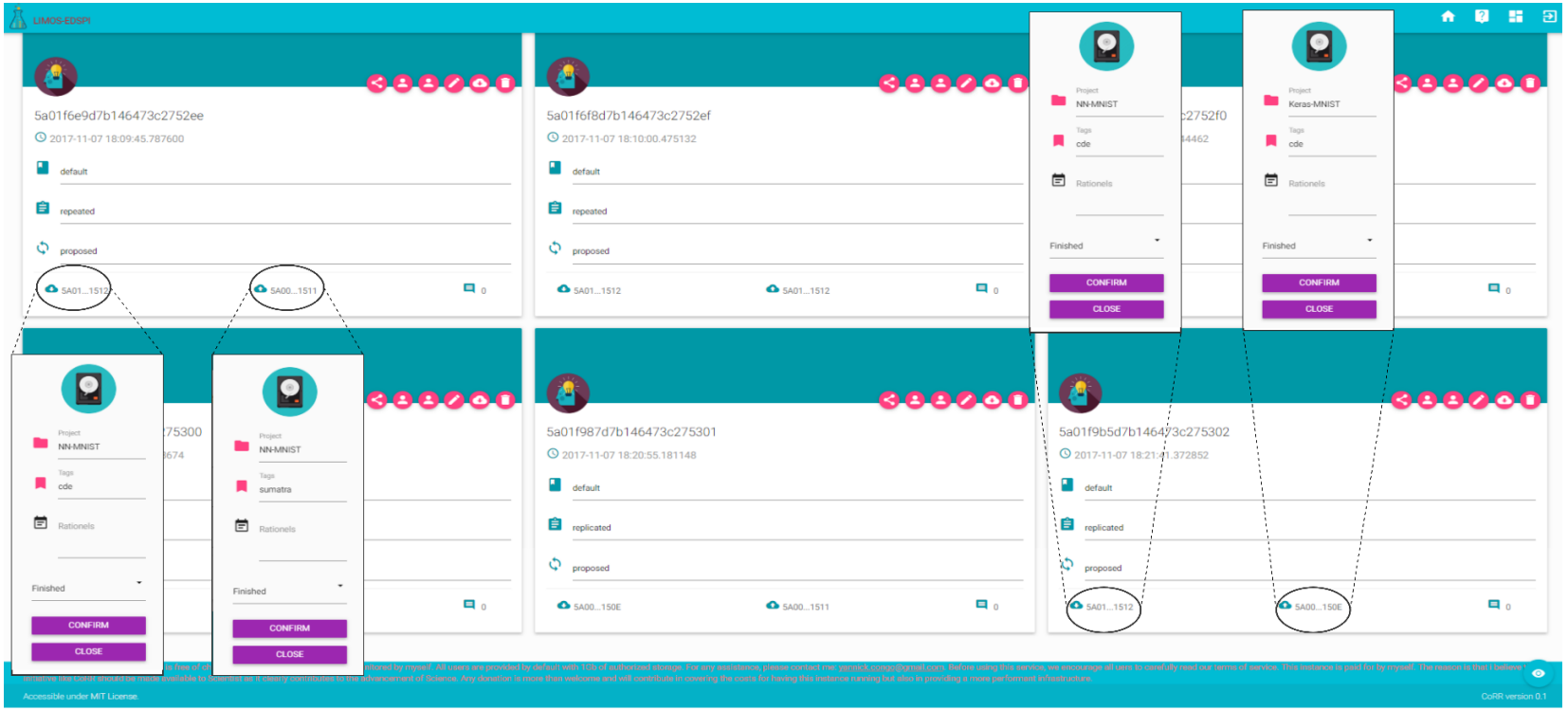

Figure 4.10 Diffs pages of Project NN-MNIST: The following figure shows the six diffs produced with NN-MNIST records. The three initial records done with each of the three tools where compared in these diffs to each other and to those of the similar project Keras-MNIST. We show the records involved in two diffs. In the first diff in the top left corner, a repeated diff has been created to signify that a record of the same computation by Sumatra or CDE are repetitions. In the second diff in the bottom right corner, a replicate diff has been created to demonstrate that despite CDE being the same tool used in the two records, there are referencing two different implementations (NN-MNIST and Keras-MNIST) that are aiming to solve the same problem, meaning obtain the same results.

4.6 INTEROPERABILITY UTILITY BETWEEN CORR, TOOLS AND PLATFORMS

Achieving interoperability between tools in CoRR took more than the comprehensive MDE detailed previously. In our considering of CoRR as being a unification point suppose that it will bridge the equivalent flow of representations from one tool to the other as shown in Figure 4.11. In the latter we draw the case of Scientist-**1** and Scientist-**2** using different tools Tool-**A** and Tool-**B** to manage their computations. When computing the same code (Experiment-**1**), the two tools produce Computation-**A1** and Computation-**A2** locally. When pushed to CoRR, the two representations take an equivalent form as Computation-**C1** and Computation-**C2**.

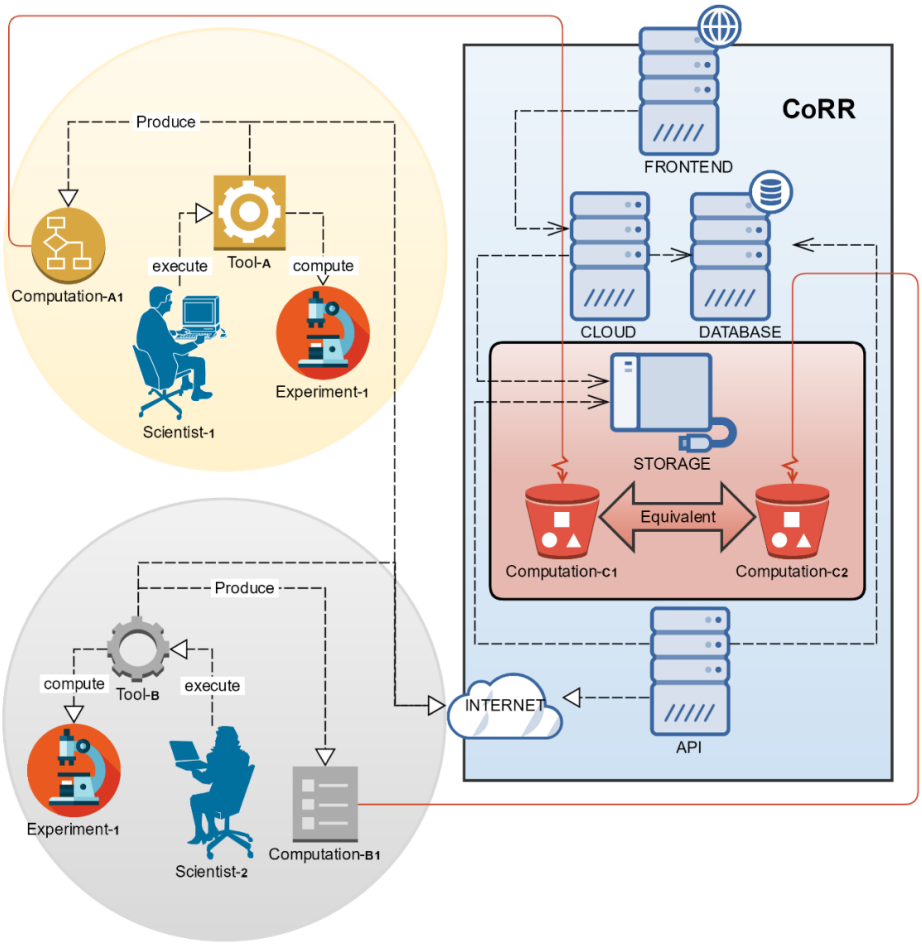

Figure 4.11 Interoperability case between Tool-A and Tool-B with CoRR

When comparing the three possible ways of representing a computation (Metadata, Hybrid data and Raw data) we have come to realize that there is an intrinsic logical flow with regards to how each of these structures can be comprehensively processed. Therefore, we have enumerated three generic state machines that capture this inherent logical flow as shown in Figure 4.12.

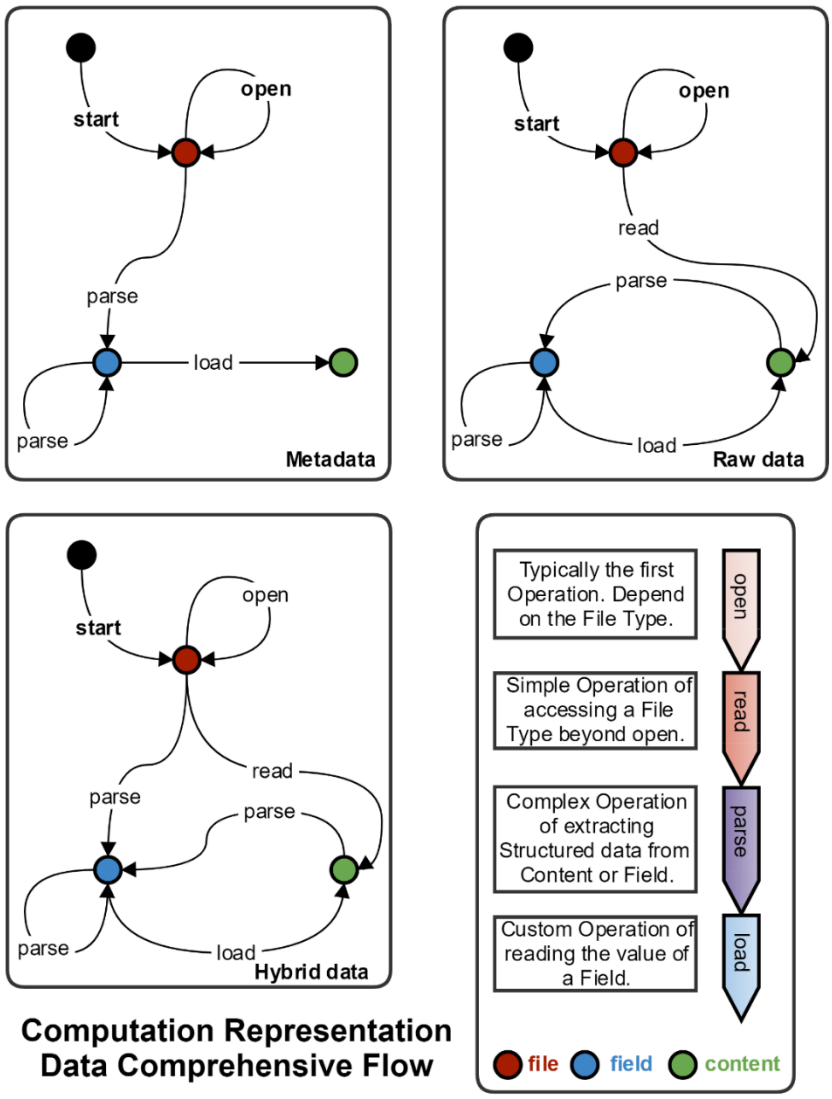

Figure 4.12 State Machines capturing the three Representation Types: Each of these state machines displays a comprehensive way to process each type of computation representation record.

From the state machines shown in Figure 4.12, we can generate an ingestion recipe that is specific to each tool. We have named these recipes “Flowers” to refer to the fact that they encapsulate the flow of operations needed to process the data. In table 4.4 we demonstrate how these state

machines can guide extract the flowers of Sumatra, ReproZip and CDE. To produce a flower, the succession of the actions is based on any list-based content being sorted alphabetically independently of its nature. This allows consistency across representations flowers when executing them. We have used the YAML as the language to store the flowers. As such it uses YAML construct to represent the succession of the operations (open, read, parse and load). Moreover, while a list node indicates that the operation will be repeated across a list of content, a field element indicates that it is an optional operation as the other fields of the same level are.

Table 4.4 Sumatra, ReproZip and CDE record processing Flowers

Tool	**Sumatra**	**ReproZip**	**CDE**
Type	Hybrid	Raw	Raw
Logic	Data: File [folder] project: File [json] record: File [json]	DATA.tar.tgz: File [folder] Hash: File [folder] DATA: File [folder] bin: File [folder] etc: File [folder] home: File [folder] lib: File [folder] lib64: File [folder] sbin: File [folder] usr: File [folder] var: File [folder]	cde-package: File [folder] cde.environment: File [raw] cde-exec: File [raw] cde-root: File [folder] lib: File [folder] home: File [folder] etc: File [folder] origin-run-pwd-txt: File [text]
Flower	start: # return list of files -open: #for each -open*: #repeat for each read: #do this open: #or this -parse*: #or repeat for each load: #do this parse: #or this	start: # return DATA.tar.tgz folder open: # return Hash folder open: # return DATA Hash folder open: # return list of files -open: -open*: #repeat for each read: #do this open: #or this	start: # return list of files -open: #for each -open*: #repeat for each read: #do this open: #or this -read: #or repeat for each parse: #or repeat for each

Sumatra flower can be read as: open the record and expect a list. Then, for each element open or parse in a loop fashion. So, after opening, read or open the file and if opened do this again. And, after parsing, load or parse and if parsed do this again. While its flower allows automation with regard to Sumatra's structure logic, it is still generic. In fact, we have not specified what the

operations open, read, parse and load are. While in the case of Sumatra we can say that they are respectively glob.blog, open, json.load(open) and any dictionary, in case of CDE, the parse operation (aimed at cde.environment and orig-run-pwd.txt) requires a custom method to extract some content.

While its flower allows automation with regard to Sumatra's structure logic, it is still generic. In fact, we have not specified what the operations open, read, parse and load are. While in the case of Sumatra we can say that are respectively glob.blog, open, json.load(open) and [], in case of CDE, the parse operation (aimed at cde.environment and orig-run-pwd.txt) requires a custom method to extract some content.

We have researched the processing recipes (flowers) for automation but beyond this aspect, we have designed them as a practical way to achieve interoperability.

In fact, when two flowers are obtained for two different tools, a mapping specification between key elements of both is enough to allow a certain degree of equivalence. To be more specific, guided with a mapping specification, a flower can construct a new record from a processed one. In Table 4.5, we provide a mapping specification between ReproZip and CDE.

Table 4.5 Interoperability mapping specification between ReproZip and CDE

Tool	**ReproZip**	**CDE**
Mapping	# hierarchy definition ?:DATA.tar.tgz # a generically named folder in contained DATA:? # folder DATA is in this generically named folder cde-root:cde-package # cde-root is contained in cde-package	
	bin:DATA bin:DATA bin:DATA etc:DATA home:DATA lib:DATA lib64:DATA	lib:cde-root cde_exec home:cde-root etc:cde-root home:cde-root lib:cde-root lib:cde-root

The reader will gather that a mapping specification is simply a file that lists equivalence between flower operations results. For example dependencies:record=lib:cde-root:cde-package means that at the moment of parse(record), load(dependencies) is equal to open(cde-package), open(cde-root).open(lib). Thus, every row in the dependencies field in Sumatra record would be file inside the lib folder in CDE.

We would like to stress that ":" is used to indicate path precedence. Furthermore, the reader must keep in mind that a mapping from one rule to the other is matching the equivalent flower operation pipeline output. Thus, only identical or close in value results in the output will be converted. For example, the two last lines indicates that from CDE standpoint the content of lib folder can be built from the content lib and lib64 folders in ReproZip. Additionally, from ReproZip standpoint, the content of lib and lib64 can be built from the one from lib in cde-root. In the latter case, some duplication will occur but will not incur in any issue.

The reader might have noticed that the provided mapping does not cover ReproZip and CDE structure logic fully. Files such as sbin, usr, var, origin-run-pwd-txt, environment.cde are not mapped. While it is not possible to guarantee the mapping of all components since some tools may not need as much information as others, it is possible to provide customized mapping functions. An example would be a mapping function "data" used in the mapping data~inputs:record=var:cde-root:cde-package to match input file names from Sumatra to CDE. The same mapping function could be applied to outputs.

The result of this section shows the ongoing exploration of a novel way to achieve interoperability between tools in a way that benefits their users. As such, it was implemented in a library named Contracts (https://github.com/usnistgov/contracts accessed September 26, 2018). While this implementation is still very experimental, we are confident that it is our assurance for reproducibility between reproducible research tools and platforms. However trivial with CoRR as we provide default components for the latter, the interoperability crafting may bypass CoRR completely.

4.7 CONCLUSION

This chapter, while emphasizing the necessity of version control tools, postulates that standard source code versioning is not sufficient for reproducible research. The chapter advocates for an alternative form of version control based on the capture of computational provenance and environment. Reproducible research support tools have the potential to become a critical component of reproducible research workflow and complement more established approaches such as Software version control. However, to achieve broad acceptability, a highly functional web platform is required to support sharing and dissemination of support tools records.

Thus, this chapter focuses on introducing the Cloud of Reproducible Records (CoRR) as a web platform for these tools. Among the features delivered by CoRR, is the enabling of a new way of scientific networking. This networking is made possible by integrating tools, allowing scientists to collaborate with each other around scientific computations and findings. CoRR also addresses the issue of properly framing reproducibility assessments. An assessment in CoRR is a one-to-one relationship between records in which a resulting decision is made to conclude if two computations are repeats, reproductions, replicates or their opposites. While automatic assessments are possible up to a certain point, it also allows scientists to contribute by either initiating the assessments or participating in the decision. A case study is reviewed to allow the reader to grasp the importance of integrating the three major tools with CoRR. Furthermore, a live instance of CoRR is made available in support of this case study and to the community. It contains the recorded computations and resulting assessments of ANN examples from Dr. Adrian Rosebrock's book.

These tools are currently not interoperable. This is a perfect fit for CoRR. First, because it designed to deliver an intermediate model that can store different tools representations of computations. Secondly, it eliminates the skepticism toward adopting any of these tools, specially knowing that the obsolescence of a tool is equivalent to future obsolete records.

Moreover, making computation records representations equivalent is not the end goal of this initiative. The ultimate outcome is to allow scientists to re-compute executions from a CoRR record. This is how reproducibility, replicability and repeatability are verified and corroborated.

Thus, tools development teams must first define a minimal set of features for a tool to be called a tool. In fact, some for example will argue that a tool should be called a support tool, when it does not allow a consistent re-execution of the its previously recorded computations. CoRR is expected to moderate these discussions.

The presented challenges are opportunities for improvement and innovation. They lay the way for CoRR's future and features. They also point the fact that CoRR's road map requires playing a central role between tools by leading the way for a sustainable ecosystem of useful and interoperable reproducibility tools. One of the most critical challenged that remained unsolved by CoRR is the actual corroborative nature of the execution context representations. Most of the current solutions cannot guarantee as exact run to run corroboration, especially in the meaning of numerical repeatability. The following chapter shows that an effective solution to this major challenge must come from deep within the computation of numerical operations. As such we propose an approach that implements such a solution.

Chapter 5 – COMPUTATION OPERATIONS CACHING FOR NUMERICAL REPEATABILITY

5.1 INTRODUCTION

We are currently witnessing the scientific method alarm buzzing across many fields of Science either may it be more of a traditional experimental one or a modern computational one. Terminology in reproducible research (repeatability, reproducibility, replicability, etc.) allows one to place himself in the appropriate context to interact within its own community. In fact, we have yet to reach a terminology consensus across domains as pointed out by many [**Drummond 2009; Gordon-McKeon 2015; Slezák and Waczulíková 2011**]. Throughout this chapter we will be using the same terminology as used previously in the present manuscript. We recall the latter as following:

Definition 5.1. Repeatability is the simple re-execution of a simulation while enforcing the preservation of the previous execution context (environment) to get the same result.

Definition 5.2. Reproducibility is any attempt to reach the same result as a previous execution, yet tolerating changes in the environment, the dependencies, the inputs, the executable itself as long as the output results stay in the meaning of the scientific background to be acceptably conform.

Definition 5.3. Replicability is any attempt within the scientific logic that leads to the exact same result. Variations are not tolerated in the results. The goal is to have an identical result no matter what was used to get there.

Sometimes as Reproducibility is the most compromising one of these notions in terms of tolerated variations, we will prefer to use it when referring to all terms in a general sense.

While a list of probable sources of issues in reproducibility, repeatability and replicability is still being investigated in various domains, there is an intuitive growing agreement that solving these problems requires the proper capture of some key elements that contributed to the computational results [**Sandve et al. 2013**]:

Definition 5.4. Environment means enough information about the system in which the simulation was run (hardware, operating system and compiler).

Definition 5.5. Dependencies represent all the elements required by the simulation code to be properly built and executed (libraries).

Definition 5.6. Inputs contains all the data read by the simulation to produce the outputs at the end of its execution.

Definition 5.7. Executable signifies enough information to get and execute the simulation (source code, binary, execution command).

Definition 5.8. Outputs refer to all the data produced by the simulation during its execution.

Recording these elements, should allow us to recreate the computational requirements needed for the executable to produce the same outputs (as recorded) when run with the same inputs. This means being able to numerically repeat a computational experiment. It is often called numerical reproducibility [**Hill 2017**]. The current representations of the software execution context presented in the previous chapter are based on this philosophy. Even CoRR.

However, we argue here that for the numerical repeatability of computational results the tracking of these key elements is not enough. The numerical result is the complex functional aggregation of all the arithmetical operations results computed by the CPU during the execution. And the currently recorded key elements do not preserve any trace of that information except the outputs which usually retain the final results. Thus, to guarantee that a result be replicable, repeatable or reproducible one will have to wait until the end of the next run to be able to compare the outputs when possible. This requires also more rigor in the structure of the outputs. And since only final numerical values are usually saved, identifying the origin of a variation is not a trivial task. A classic example would be the impact of out of order execution on floating point operations [**Zitzlsberger 2014**] and the level of difficulty involved in debugging such a situation.

We propose in this chapter to introduce another key element: the numerical computations cache. It is the capture of all the mathematical calculations done during the execution. Thus, during another execution of the computation, this cache can be of great use when attempting to

numerically repeat the result. It helps to track at a deeper level the source of the numerical variations and apply various execution strategies which is not possible with the currently referenced key elements. Furthermore, every cache entry name is strongly correlated to the symbolic representation of its corresponding computation and is time stamped. This makes the construct sensitive to precision changes and the execution order of the underlying computations that produced the operands fed in. Therefore, any variations in these two cases due to facts like non-associativity in Q [**Goldberg 1991**], will either lead to a detected result variation or to the creation of a new cache entry and by the same token cause some inconsistencies in the cache. In Table 5.1, we demonstrate trivial cases in which the loss of the associativity for the core mathematical operators (+, -, *, /) is obvious.

The proof of loss of associativity shown here can only be worse for complex functions which are composed of these core operations. In fact, the resulting precision errors will be accumulated and thus cause major drifts in the final results.

In this chapter, we first overviewed the current techniques used to guarantee numerical reproducibility in a more general sense (including replicability and repeatability), then we will go into more details in our proposition with its drawbacks and finally provide an application use case.

Table 5.1 Non-associativity demonstration for the four core mathematical operators

Addition			**subtraction**			**multiplication**			**division**		
a	**b**	**c**	**a**	**b**	**c**	**a**	**b**	**c**	**A**	**b**	**c**
0.001	1	-1	1	1	0.001	0.001	1e308	11	100	1e308	11
(a + b) + c			**(a - b) - c**			**(a * b) * c**			**(a / b) / c**		
0.00099...86588			-0.00100...00208			1.09999...99722e+306			9.09090...13424e-308		
a + (b + c)			**a - (b + c)**			**a * (b * c)**			**a / (b * c)**		
0.00100...00208			-0.00099...86588			inf			0.00000...00000e+00		

5.2 NUMERICAL COMPUTATION CACHING

During the process of implementing a computational solution, debugging is the most painful step. This is especially due to all the sources of irreproducibility that can cause a simulation to give different results at different moments in time with the same or different computing environments. Profiling and debugging tools like prof, Valgrind, Dr. Memory, Jtracer, Vtune… are typically used in this case to track the simulation execution as a process in mostly the aspect of memory management and threading. We argue that these techniques and tools do not provide any persistent result from their integration that can be reused later as a mean to ease repeatability or to investigate numerical reproducibility. We present in this research, the investigation of a numerical library Num-Cache that when integrated into a simulation code, can be setup to generate the full numerical cache of the scoped computations as shown in Figure 5.1. The caching mechanism provides interesting features that will be discussed in the next section.

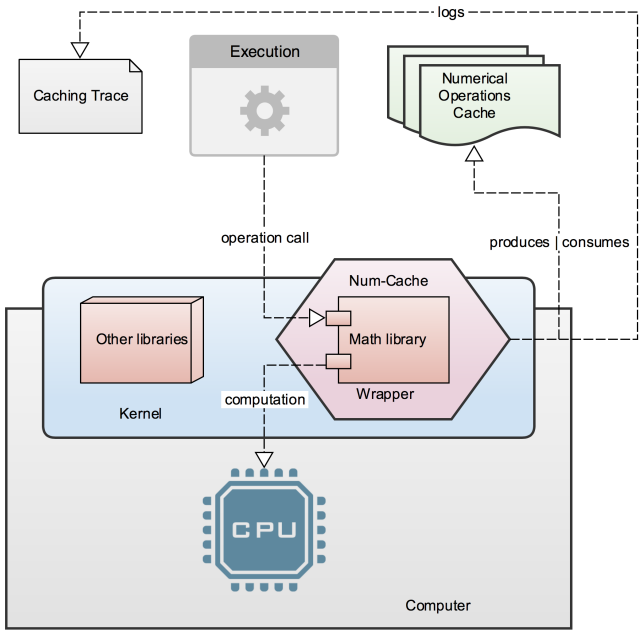

Figure 5.1 The Num-Cache functional architecture: The mathematical operations calls go through the library first, which generates the cache entries after the actual computation by the CPU.

5.2.1 Mathematical operation caching

In most programming languages, the mathematical functions provided for calculus are often split into two libraries: the standard library and the maths library. The standard library provides operators like -, +, * and /, while the maths library provides approximated evaluation of more complex functions like exp, log, log10, pow, sqrt, etc. All the operators and functions can be roughly grouped based on the size of their operands in three classes that we name here: **0-Operand**, **1-Operand** and **2-Operands**. A **0-Operand** operator is in fact a function called like procedure with no parameter: a call to a basic rand function is an example. A **1-Operand** operator (monadic operator) is an operator or a function requiring only one operand or parameter: like the negative '-', exp, log, sin, etc. And finally, a **2-Operand** operator (dyadic operator) requires two operands: +, -, * and /. While it is possible to have more operands, in the case of this research we limit the number of any operator parameters to two at most. We propose to atomically and uniquely cache every call to these operators by capturing: information about the operator, the result and the operands.

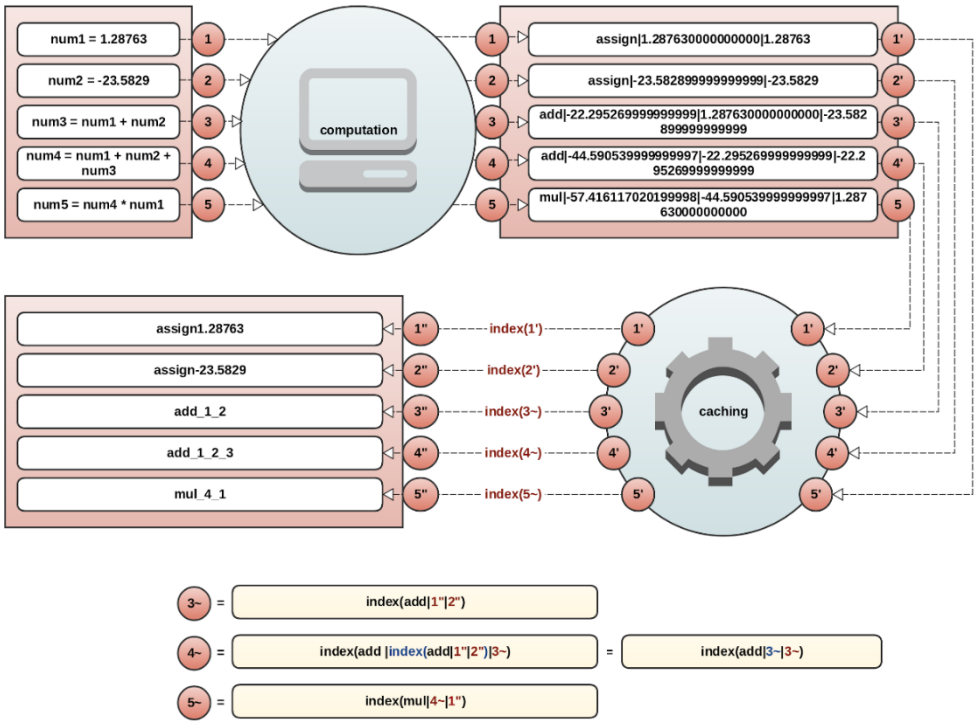

Figure 5.2 The computations cache generation: It is demonstrated on an example involving 5 computations.

In this investigation, we assume assignment operator calls as root operations. This assumption is justified by the fact that all other operator calls are finalized by assignments to variables. It is also required as all the mathematical operators are subjected to cause variability in the outcome. Moreover, we want to separate precision errors which usually come during assignments from out of order execution errors and cumulative addition in limited precision. Thus, we propose an indexing technique that uniquely identifies the cache entries as shown in Figure 5.2.

This is done here by generating the index of any computation from its content by replacing its operands values by the index of the computation that produced them. This rule applies for all operators except from the assignment. This indexing ensures that the mathematical relations between the computations be preserved. This indexing mechanism ensures also that the link between the computations be sensitive to precision error during assignment and from out of order execution during any other computation. Thus, it starts with the assignment's operations cache entries. To do so, it generates index strings from the content of the assignment's caches (operator, operands).

Since we want to preserve the relationships of the mathematical operations, we cannot generate their cache contents like the assignments. Because, the generation from the content of other operations will lose the relationship nature among them. Therefore, the assignment is assumed to be a root operator. The operands of an operator are either results from an assignment or from another operation. Thus, we define the index of the other mathematical operators as being based on the indexes of operations that produced their operands (assignments or others). This provides a chain of indexes that from any mathematical computation cache folds down following the computations relationships to the assignment indexes (as roots). Precision errors will be detected early on from loading the assignment operations and comparing the results. Out of order errors on the other end will be detected by a change in the relationship order between operations. This modification will then make it impossible to properly reload the appropriate computation and make sustainable comparisons. It is caused by the fact that we generated the index of all computations of operators based on their mathematical relationships to others through their operands. In the advent of a different result within a single operation we can track the impact of

this modification through the solid mathematical relationship graph between operations. Also, this indexing technique allows us to always guarantee identical cache entries even when the computation result varies for non-assignment operations, which is fundamental to be able to reliably compare two caches that may have different results that are not caused by precision error in the same cache entries. In addition, this chaining between computations through their operands allows us to first detect precision errors for assignments; and then to identify execution order variations for other operators. The occurrence of these variations will trigger the generation of new indexes and will be propagated throughout the chain of computations. As such, an inconsistent cache will be generated. When reloading the numerical cache of a previous execution we propose three strategies: ignore-cache, use-cache and load-cache. When using ignore-cache, the library generates a verbose log of the comparison between the loaded cache computations results and their current computations results. This allows the detection of precision errors, out of order computation, their impact and individual operations results changes. On the other end, use-cache will simply load the cached results when the newly computed ones are different and log their variations. Finally load-cache directly loads all the results of the operations without any logs of the variations and in summary not doing the actual computation on the machine.

5.2.2 The Num-Cache library

We propose the implementation of the Num-Cache library for C++ and Python. The Num-Cache library overloads selected mathematical operators and functions. The cache content is: operator|result|operand1|operand2 for 2-Operand operators, operator|result|operand for 1-Operand operators and operator|result for 0-Operand operators. For every cache entry the name is the index of the content of the cache without the character '|' and the result. The source code is online can be found online on Github at the following url : (**https://github.com/faical-yannick-congo/Num-Cache** accessed September 29, 2018).

The repository contains a proof of concept application folder that will be explained in the next section, a cpp and python folders that hold the implementation of the Num-Cache for the two

supported programming languages. Test programs in C++ and Python are provided and are the numerical caching of the following block of pseudocode in Table 5.2.

The Num-Cache library comes with a Numb Entity that is used to wrap assignment values and overload the operators.

Table 5.2 The two ways "a+b+c" can be evaluated

Left-add	Right-add
a = 0.001	a = 0.001
b = 1	b = 1
c = -1	c = -1
d = (a + b) + c	d = a + (b + c)

The Num-Cache library should be initialized before the run using a setup function which takes four parameters: cache_out, cache_in, precision and strategy:

Definition 5.9. cache_out is the produced cache location.

Definition 5.10. cache_in is the path to a cache to load.

Definition 5.11. precision is the precision of the computations.

Definition 5.12. strategy is the input strategy cache (ignore, use or load).

For testing, we first do the numerical caching of left-add without an input cache at a precision of 20 and generate 'cache1'. Then we do another numerical caching of right-add by loading the previously generated cache 'cache1' but using the ignore-cache strategy. This produces the Num-Cache comparison log and the second cache 'cache2'. The Num-Cache implementation for both Python and C++ is based on floating point representation. As a result, with the C++ and Python implementations, the execution of the test produces two caches: cache1 and cache2. The output of Num-Cache shows different caches. Moreover, the execution of the test in both C++ and

Python gives as expected different caches as soon as the addition operation begin to be evaluated as shown in Table 5.3 at entry orders 4 and 5.

Table 5.3 Computation cache entries for (a+b)+c and a+(b+c)

Entry order	Entry cache name for (a+b)+c	Entry cache name for a+(b+c)
1	assign0.001	assign0.001
2	assign1	assign1
3	assign-1	assign-1
4	add_1_2	add_2_3
5	add_4_3	add_1_4

The assignment of a value or a computation result to a variable is identified as the assign operator when cached. In this example, each corresponding cache entry contents for left-add and right-add are compared and shown in Table 5.4.

Each implementation of left-add and right-add in Python and C++ return the same cache. The entries shown in Table 5.3 allows us to compare the content of the cache1 (from running left-add) and cache (from running right-add). They start to differ as pointed in Table 5.4 for entry orders 4, 5 (left-add) and 4', 5 (right-add). This clearly shows that Num-Cache caching technique is able to single out operations in which the execution order change for some reason. Thus (a+b)+c is different from a+(b+c) for the specified values. Further details with more operations is provided in the use case section.

Table 5.4 Computation cache entries contents

Entry order	Entry cache content (operator|result|operands..)
4 [(a+b)]	add|1.00099999999999988987|0.0010000000000000002|1.0000000000000000000
5 [(a+b)+c]	add|0.00099999999999988987|1.00099999999999988987|-1.0000000000000000000
4' [(b+c)]	add|0.0000000000000000000|1.0000000000000000000|-1.0000000000000000000
5' [a+(b+c)]	add|0.0010000000000000002|0.0010000000000000002|0.0000000000000000000

5.2.3 The computational costs involved

The method proposed here has the drawback of lowering the computation speed as one would expect. Thus, scaling up is consequently problematic. Following are the extra operations added to the executed simulation for each strategy in Table 5.5. By integrating the proposed approach library, one should expect a multiplicative complexity factor of O(log n). This is interesting for applications that cannot afford to have a non-numerically debuggable result.

Table 5.5 Computation cache entries contents

	indexing	loading	comparing	assigning	loging	Total
use-cache	O(1)	O(log n)	O(1)	O(1)	O(1)	O(log n)
load-cache	O(1)	O(log n)	Ø	O(1)	O(1)	O(log n)
ignore-cache	O(1)	O(log n)	O(1)	Ø	O(1)	O(log n)

We also stress the fact that the library is implemented in the scope of a centralized cache. For distributed computation complicated situations might come up. Distributed high performance computing databases might be a solution.

5.3 USE CASE

In Table 5.6 we show the use of Num-Cache on the four codes provided in Table 5.1: **(0.001+1)-1**, **(1-1)-0.001**, **(0.001*1e308)*11** and **(100/1e308)/11**. Here, for comparison purposes, we have generated a SHA-256 signature of each cache entry for each operator on steps 4, 5, 4' and 5' as shown in Table 4 for addition. The differences between the signatures of the two operations involved in (a op b) op c and a op (b op c) demonstrate how Num-Cache detects out of order discrepancies. Thus, running Num-Cache with a computations cache from a system which evaluates [a op b op c] as [(a op b) op c] will throw an out of order alert on a system which evaluates it as [a op (b op c)].

Table 5.6 Signatures of the two computations steps for each operations factorization

signature generation	addition	Subtraction	multiplication	division
(a op b) op c	**(a + b) + c**	**(a - b) - c**	**(a * b) * c**	**(a / b) / c**
signature {4: op_1_2}	710f...de22	b39c...d55b	2d35...75a8	b39c...d55b
signature {5: op_4_3}	fd7e...f277	4dfc...2554	6110...35a1	309b...6672
a op (b op c)	**a + (b + c)**	**a - (b + c)**	**a * (b * c)**	**a / (b * c)**
signature {4': op_2_3}	fbd3...14ad	1fb2...43e5	d432...9955	d432...9955
signature {5': op_1_4'}	4535...69a5	4f6b...8a7d	41c6...cc47	010d...378e

Num-Cache goes beyond simply caching. Its cache entry keys are the operations expressions in their order of computation. Also, each cache entry contains the operation result and operands. Therefore, when a new signature is created in a future computation, it implies that there was a change in the order of the two operands with regards to the operator. This, indeed, because the main purpose of the signature is to capture this fundamental relationship. Thus, the creation of new cache entry keys confidently indicates an out of order computation as shown in table 5.6.

Furthermore, when the signature of a new computation is identical to the one of a previous one, Num-Cache will carry on to the comparison of their results. If they are different, it is this time not an issue in the order of the operations but a failure/error in the operator execution itself. While the latter issue is rarer in ordinary computers, in exa-scale machines they appear more frequently. These two features allow the separation of issues due to pure out of order execution from pure operation inaccuracy or fault. Finally, a byproduct with having a previous cache is that Num-Cache can be tasked to trust the new or the cached result and thus achieve true numerical repeatability while alerting when issues are detected and avoided. In fact, with Num-Cache, scientists can identify which portion of their applications to watch closely for numerical changes. Also, in case of a change, the library can accept the new result or load one of a provided cached computation or just crash. In any case, a computation log is generated by the library to show when issues occur, and the strategy taken for a deeper investigation.

Despite its interesting features, Num-Cache must be used properly for maximum adequacy. For static executions, where all the possible computations are done for any execution, the Num-Cache code works out of the box. Yet, for dynamic executions where the number of operations and the values on which the computations are done varies from one run to another, the best caching results are obtained when the computations domain is well covered, which implies generating as much cache as possible so that Num-Cache can load it back later on, as shown in Code 5.1.

```
try:
        from sklearn import datasets, svm, metrics
        deep = true, digits = datasets.load_digits()
except:
        from tensorflow import tf
        from tensorflow.examples.tutorials.mnist.mnist as digits
        deep = false
images_and_labels = list(zip(digits.images, digits.target))
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
if deep:
  x = tf.placeholder(tf.float32, shape=[None, 784])
  y_ = tf.placeholder(tf.float32, shape=[None, 10])
  W = tf.Variable(tf.zeros([784,10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.nn.softmax(tf.matmul(x,W) + b)
  cross_entropy = tf.reduce_mean(
                                  -tf.reduce_sum(
                                                  y_ * tf.log(y),
                                                  reduction_indices=[1]
                                                 )
                                 )
  sess = tf.Session()
  init = tf.initialize_all_variables()
  sess.run(init)
  training = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  for i in range(4001):
    sess.run(training, feed_dict={x: data[:n_samples // 2], y_: digits.target[:n
_samples // 2]})
else:
  classifier = svm.SVC(gamma=0.001)
  classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
```

Code 5.1 Dynamic execution of machine learning libraries on the MNIST dataset

In this example in Code 5.1, we propose to the reader to consider a Num-Cache wrapped version of sklearn and tensorflow. They are two machine learning packages in Python. We demonstrate the case of dynamic execution and the limitation of the Num-Cache approach with regards to such a situation. When executing the code provided, if sklearn is available it will launch the C-Support Vector Classification (SVC) algorithm. Otherwise if not, we suppose that tensorflow must be present. Thus, we launch an Optimized Gradient Descent algorithm. Both algorithms are used to produce a trained machined learning model that can recognize handwritten characters based on the MNIST dataset. The accuracy of the first algorithm is between 0.93 to 1.0 while the second is in average 0.9 for 4000 steps.

In this example, we invite the reader to consider it as a process of trying the recognition with the best algorithm and that in the advent of it being unavailable, we use the less accurate option. Moreover, while the import calls may not cause any variation to the result, the operations involved in both algorithms are different. Thus, two possible cache can be generated from this code. To ensure numerical repeatability, the scientist must generate and use the merge cache of these two cases. In fact, if not done so, Num-Cache will simply not be capable to check for numerical variations as it may be faced with unseen operations from the other algorithm.

5.4 CONCLUSION

We proposed in this chapter, an additional key element that is important to be recorded as part of the data produced when running a simulation. The current approaches provide a form of an approximation of either intermediate or final results within a certain error margin. Also, they do not allow an easy debugging of the internal numerical variations. In this chapter, we described a numerical caching approach that enables true numerical repeatability when using load-cache as a caching strategy but also logs all the numerical operations variations during the execution of the simulation when using ignore-cache as a strategy. Another caching strategy use-cache is similar to load-cache yet only logs the variations and uses the cached result instead of the newly calculated one. We scoped the features that Num-Cache encapsulated as part of our interest to approach numerical reproducibility. Also, we have achieved implementations in both C++ and Python. Demonstration codes covering the widely used operators (+, - , *, /) were also

implemented in both C++ and Python. By focusing the issues of non-associativity due to a change in execution order for these core operators, we are able to press the greater presence of the issues in more complex functions.

Num-Cache is designed with an internal representation in floating point only as most problems occurs with these. As a proof of concept, the library only covers the most used mathematical operators. In order to build a reliable cache index that may change only in specific situations, we have made the assumption that the assignment operator is a root operator. The caching process is centralized and only captures executed operations. For executions that might trigger the computation of other operations for other runs, we recommend as much computations operations as possible to allow the entries to be created in the first run. This research and the current state of Num-Cache does not include any symbolic execution capabilities and involve complexity costs in the order of O(log n) across the three strategies.

It is fair to ask the reproducible nature of a method that is attempting to enforce reproducibility. As a matter of fact, Num-Cache is a wrapper that mostly persists its cache and uses it based on the described strategies. A reproducibility issue may occur if the cache is corrupted. In fact, tampering with the cache in between runs or during the run due to a file system or a database issue will most likely lead to the skipping of the Num-Cache execution.

The evolution of computer hardware architectures and their diversity is bringing an awareness regarding the non-portability of software execution (numerical irreproducibility) [**NRE 2015**]. As depicted by V.T. Dao and his colleagues [**Dao et al. 2014**] through a survey on different hardwares, operating systems and configurations, we have to better grasp the complexity of debugging for numerical irreproducibility as it is a problem now and still an expected one at Exascale.

Being able to complement the computation execution context represented by all currently existing tools and platforms, the Num-Cache approach, allows CoRR to reach the ultimate representation model. Such model can effectively augment the current tools representations in all three corroboration levels.

However, one major drawback of the cache based approached used in Num-Cache is the toll incurred on computation intensive applications. Despite the use of a hash table-based storage strategy, the overall computing time decrease will clearly be noticeable. Moreover, during the re-execution, depending on the strategy used, additional computing time will incur. In all cases where the cache is not directly reloaded such a slowdown will be present. In the case of full consideration of the cached computation, many factors must be considered. In fact, if the storage media can match computation time compared reloading the result from the hash table then no computation time will incur. Based on this evident ongoing challenge in our approach here, Num-Cache will be mostly useful in situations such as:

- Debug phase: When building the scientific code, it will help track the numerical aspect.
- Few operations: When the experiment does more than just numerical operations.
- Custom caching: Identify weak numerical places (big sums, etc..) and only cache those.

In the following chapter, we invite the reader into a brief tour of the most substantial applications of this thesis. We first present the current work in integrating major reproducibility tools and their representations with CoRR. Then, we share actual use cases in which either CoRR is involved directly or the episteme and research knowledge collected during this journey was solicited.

Chapter 6 – APPLICATIONS

6.1 INTRODUCTION

The umbrella of activities done during this thesis are still ongoing and mostly involves CoRR. These activities are organized in two groups. The first group focuses in leveraging CoRR's main feature in solving $\mathbf{P_2}$. As such, it is composed of the current integration work with existing general-purpose reproducible research tools. The second group presents activities that demonstrate the usage of CoRR in ongoing projects. We list three of the most relevant ones at NIST and two others with external collaborators. The following subsections give more details to these activities.

6.2 INTEGRATION ACTIVITIES

One of the main features of CoRR is its capability to integrate any tools or web platforms. This is possible because of its open API access and the flexibility of its storage model. Thus, tools and web platforms integrated to CoRR can allow scientists to interoperate seamlessly. In fact, within the CoRR interface, all records are homogenous despite the specifics of the execution record being heterogenous to one another depending on the tool. Currently, CoRR supports three of the most known general-purpose reproducible research tools. The following subsections describe each of those.

6.2.1 CNRS - Sumatra

In this integration with CoRR, scientists using Sumatra are able to push meta-data, inputs and outputs files of their simulation's executions to CoRR. This integration comes as an additional feature of the tool. Thus, it does not eliminate the default storage capabilities. Instead it coexists with those. Therefore, when recording a simulation execution two records are created. One local where Sumatra has been setup to record it and another one remotely on the configured CoRR instance. The current integration only supports submitting records in one direction: from Sumatra to CoRR. All other actions are not currently available from the tools and must be done within CoRR web interface: Pull, Edit, Deletion and Collaboration. Future integrations include extension to pull, edit, delete and collaborate from Sumatra Command line through CoRR. The

integration to CoRR is currently held in a forked copy of the official Sumatra code and is still in experimental phase. It is accessible at: (**https://github.com/usnistgov/corr-sumatra** accessed September 29, 2018). It is open source and we look forward for an integration level that will meet official acceptance.

6.2.2 NYU - ReproZip

When integrating this tool, we had a version that was not exposing the meta-data construction in an open fashion. As such, the integration is currently limited to a simple push of the 'rpz' compressed file that represents the execution. However, this situation has evolved. The current version of ReproZip fully details the meta-data captured. Thus, future integrations will involve a more complex integration that is similar to what is proposed for Sumatra. Also, since ReproZip records can be turned into Docker containers or other similar containing structures, we are also considering a docker based pull from CoRR or a more advanced way of partially performing operations of ReproUnzip within the CoRR instance. This extends the choices of the structure to pull from: chroot environment, virtual machine and docker. The current integration of ReproZip to CoRR is also held in a forked copy on Github: (**https://github.com/usnistgov/corr-reprozip** accessed September 29, 2018). A more stable integration will be submitted for official acceptance and will give its users the possibility to link their tool to any CoRR instance of their liking.

6.2.3 MIT - CDE

Designed to directly generate a portable Linux environment that encapsulated the snapshot of the execution, CDE produces a single compressed file that is pushed to CoRR in a single API call. This integration is similar to the current one with a previous version of ReproZip. Regardless of CDE moving in the direction of ReproZip and Sumatra to add meta-data capture, we are looking for an integration in which CDE can also be used to pull records. In fact, the current integration only supports pushed to CoRR. However, if more features are enabled in the future, its integration to CoRR can be expanded for more complex interactions as done with Sumatra. The source code of the integration is also a modified version of the official code and is to be considered experimental as the two previous ones: (**https://github.com/usnistgov/corr-CDE** accessed September 29, 2018). We are also looking forward to reaching an agreement with the

tool contributors and share experiences with other tools with more complex interactions' schemes with CoRR.

6.3 COLLABORATIONS

By providing viable solutions to $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$, CoRR has received vivid interests for immediate applications at NIST. Moreover, it has stirred up collaborations with Air Force Research Laboratory (AFRL) and LLNL. Most of the applications of CoRR in ongoing projects look for leveraging its main feature as a federating interface that deliver features that are tools and web platform agnostic with regards to reproducible research efforts. As such while the tools and platforms evolve separately, CoRR adjusts to them by accordingly updating their integrations, its API and its storage model. Therefore, these platforms need not to worry but to keep up with the changes in CoRR only in the cases where the interactions protocol has drastically changed. We have designed the platform to avoid such cases and mainly focus on smooth updates with low consequences to backward compatibility. The following subsections describe these activities in more details.

6.3.1 NIST - Joint Automated Repository for Various Integrated Simulations (JARVIS) reproducible calculations with Sumatra and CoRR

With his mission to automate materials discovery using classical force-field, density functional theory, machine learning calculations and experiments, Dr. Choudhary has been leading JARVIS for a few years now. The novelty of his approach is using formal tools in automated fashion and new prediction techniques to capture the intricated existence of materials. As such, he automatically launches thousands of Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) and Vienna Ab initio Simulation Package (VASP) computations to respectively achieve force-field calculations on DFT geometries and 3D-bulk, single layer (2D), nanowire (1D) and molecular (0D) systems. The resulting data contains materials properties such as: energetics, elastic constants, surface energies, defect formations energies, phonon frequencies, diffraction pattern, radial distribution function, band-structure, density of states, carrier effective mass, temperature, carrier concentration dependent thermoelectric properties and gamma-point

phonons. Dr. Choudhary then uses the calculations done with DFT to train a machine learning model. The goal of such an approach is to be able to predict energetics, heat of formation, GGA/METAGGA bandgaps, bulk and shear modulus.

To be able to reproduce the thousands of computations done with LAMMPS and VASP, JARVIS automated calculations used Sumatra at its early stage. Moreover, the version of Sumatra used by Dr. Choudhary is the modified version of Sumatra integrated to CoRR. The execution captured by Sumatra were stored on an internal instance of CoRR at NIST. Therefore, reference of this usage of CoRR for JARVIS-DFT has been made by Dr. Choudhary in a Nature Scientific Data publication in May 2018 (Choudhary and al. 2018). The database and features of JARVIS-(DFT|FF|ML) can be accessed at: (**https://ctcms.nist.gov/~knc6/JARVIS.html** accessed September 29, 2018).

6.3.2 NIST - CHiMAD Benchmark computations in CoRR

CHiMAD is program funded by NIST to promote the excellence in the development of advanced materials through the next generation of computational tools, databases and experimental data. The ultimate goal of this program is well scoped in NIST mission to accelerate the design of novel materials and their integration to industry. Within the same program, NIST funded the PFHUB: Phase Field Community Hub website. It contains phase field benchmark problems vetted by the community to test, validate and verify phase field codes. From May to August, we have supervised graduate student Andrey Moskalenko in an internship with the mission to capture execution of phased field codes computations on some of the benchmarks on PFHUB. Like JARVIS, the version of Sumatra used by Andrey is an integrated one with CoRR. Thus, the captures where stored on the NIST internal instance of CoRR. Andrey's work was part the test phase of the earliest integration stage of Sumatra to CoRR. A result of 38 computations were captured. The outcome of the computation of these codes was mostly performance-based data: memory usage, cpu usage, duration and the actual benchmark results. While some of the computations took a few minutes, other took days and some even weeks. This was part of our goal in testing the stability of the integration for fast running computations and gradually slower ones. The PFHUB site can be found at: (**https://pages.nist.gov/pfhub/** accessed September 29, 2018).

6.3.3 NIST - FACT

Dr. Espinal at the NIST FACT lab solicited our help in addressing a possible uncertainty issue that occurred at multiple instance with the adsorption machines used. Composed of three machines, Dr. Espinal and her team have been running cycles for years now and have consistently noted some discrepancies. Despite the differences in procedures, all adsorption data could be aligned on the same sets of units. While measurements between machines were odd from time to time, what was more interesting is that the changes were occurring within the same machines. Our collaboration can be comprehended as a forward lookup into an important additional feature of CoRR. In fact, once the execution of computations is captured, one could imagine automatic mechanisms of characterizing the output through a reference structure identification. Thus, results drifting from what is known can be quickly identified and addressed. However, in the case of the FACT lab, we are in an experimental use case. The additional challenge was the investigation of a design architecture that could prove the level agnosticism that could be handled by CoRR. The result of this collaboration is a separate web application named laura-reference that can take any sorption data and whenever needed generate a reference data from all the existing datasets. As such, another feature of laura-reference is the capability to assess new datasets with this reference data. When measurements are drifting too much in some portions, the new dataset is rejected. Yet, when it is closed to the dataset in a way that is similar in parts to existing and accepted measurements, the dataset is labeled as a valid characterization measurement output. We have learned enormously from this project. First, by being a computational solution, CoRR interface can be adapted to tools and web platforms. Moreover, the software controlling experimental machines can be modified as done with tools and web platforms to communicate with an instance of CoRR. Thus, we have come to the conclusion that integrating a tool to CoRR to ensure the reproducibility of its managed computation is at a level of flexibility that its agnostic to whether the end unit is a standard computer or more of a complex custom machine. Second, the design of laura-reference has open the perspective for how plugins for extended features of CoRR could leverage the API in new ways to provide advanced mechanisms such as here: reference output fitting and output validity assessment from a large

collection of previous ones. The source code laura-reference is open source and can be found at: (**https://github.com/faical-yannick-congo/laura-reference** accessed September 29, 2018).

6.3.4 AFRL/NIST - Integrated Computational Environment (ICE)

Matthew Jacobsen and his team at AFRL have been working on the ICE project for a few years. As a data management tool that have received positive returns from its major user base at AFRL, ICE has evolved to become two entities. The original entity (ICE) core mechanism has been extracted as an independent part and named HyperThought. Therefore, ICE became the web-based wrapping around HyperThought which can accommodate various interface integrations. As part of its mission to collect data about scientific processes HyperThought inevitably came to the question of reproducibility. Thus, our Collaboration with Matthew and his team was on an integration scheme between CoRR and HyperThought that will allow any other interfaces such as ICE to leverage all the features of CoRR. A critical one of these features, is the fact that scientists can use any tool of their liking that is integrated to CoRR without having HyperThought deal with this aspect for each tool or web platform. Active discussions and meetings have started since 2017 and future work is expected later this year or earlier next year.

6.3.5 LLNL/NIST - Open Interoperability Project between CoRR and Tools

From an integration collaboration with two of the MaestroWF project's team members at a Hackaton organized by Matthew Jacobsen at AFRL, we have experienced the YAS (Yet Another Something). In this case, the Something is a simulation management package with the goal to ease reproducibility. Before this collaboration, when integrating tools to CoRR (Sumatra, ReproZip, CDE, NoWorkflow, PANDA), we were modifying the source code of these tools directly. As a consequence, we have inevitably created duplicated codes. This collaboration has allowed a step back and a reflection into how to avoid this situation. The result of this brainstorm is a shared library that will hold most common aspect of integration to CoRR. Tools and Web platforms integrations to CoRR will need this library and provide the specifics of each implementation. As a result, we are one step further into a dialog between tools and platforms development teams and closer to opening more standards in interoperability between those. The resulting library is called Contracts as it provides the means to interact with CoRR. Those means can be called in the

third parties' software in the needed order. An initial Python version is available on Github: (**https://github.com/usnistgov/contracts** accessed September 29, 2018). Binders to other language will be provided going forward.

6.4 CONCLUSION

In this chapter we have enumerated the most substantial projects applying the results of this thesis. Far from us to ignore the other collaborations not listed here. Moreover, collaborations involving our computational results caching for exact repeatability are barely starting. We expect it to be fruitful in the upcoming year. Additionally, despite most applications being on CoRR, we have a wider range of possibilities that extend beyond computational reproducibility as shown with the FACT lab. Also, dawning projects not listed here demonstrate applicability of CoRR in Artificial Intelligence, Blockchain and IoT. All of this in service to augmenting the current practices of safeguarding the trust and essence of Science. We briefly describe our integration capability with major tools: Sumatra, ReproZip, CDE. As we write this thesis, we are working on adding tools such as PANDA, NoWorkflow, MaestroWF. The reader will notice in this chapter that most use case applications are done within NIST. The reason is that as of now, there are two active instances of CoRR. One internal within NIST is only accessible to Scientists of the institute. A second instance **https://corr-root.org** (corr-root), is a production ready public and serves as an experimental instance for testing the latest releases of CoRR. We are currently working to have the first official public instance of CoRR hosted at NIST. We encourage scientists interested in using the applications to wait for instance **https://corr.nist.org** (corr.nist). We hope that by the time this manuscript reaches the public eye, the instance will be available. While corr-root will be more up to date with the CoRR source code latest stable features and will serve as a test bed for institution looking to vet new features before upgrading their instances of CoRR, corr.nist will always show the most stable and secure instance due to more regulatory applications of the Institution's Office of Information Security and Management (OISM).

Chapter 7 – DISCUSSIONS

7.1 INTRODUCTION

At the first year of this journey we have set off to uncover how we could harmonize the current landscape of ad hoc characterization of how to represent software in execution in order to guarantee reproducibility of such a state. This adventure quickly led us to enumerate the four major problems impeding the emancipation of current tools and web platforms. To quickly recall these problems, we list them here for the reader. For more details refer to chapter 2.

- **P_1.** Lack of adequate means of reaching out to Scientists
- **P_2**. Absence of a standard or an interoperability feature between the current tools
- **P_3.** Scarcity of collaborative features in a collaboration driven world
- **P_4**. No solution to exact reproducibility due to Numerical precision issues

Once determined our journey has been since then to uniformly address them. This inquiry resulted in CoRR and NumCache. These contributions effectively solve these four problems. However not in their final forms, we have done extensive research and have implemented versions of these and applied them in situations that can only but show the length of applicability of our approaches and tangible results.

Despite the great promises and effective resolution of the problems of interest in this thesis, there are aspects of reproducibility, vital to Science, that are left unsolved or might cause confusion if left unchecked. Moreover, there are scales at which our solutions have not been tested at yet and cannot be guaranteed to be viable. The following subsections briefly discuss those aspects, not solved by this research but still represents a fundamental barrier to advancing Science, its trust and collaboration features from within.

7.2 REPRODUCIBILITY VS CORRECTNESS

In our sense it is fundamental for the reader to grasp the subtle difference between these two terms. The research and solutions investigated during this thesis focus solely on the reproducibility factor of research results. By no means do we suppose correctness. By focusing

in reproducibility, our methods and all the tools presented here attempt to answer the question: given some meta-data and data structured in a comprehensive way, can I reconstruct a research computation to reach a result similar, if not identical, to the one from which those meta-data and data were taken from? This does not mean in no measure that the result obtain is correct. Consequently, a scientific investigation can be made a hundred percent reproducible but be a hundred percent wrong. While verifying the correctness of scientific results requires reproducibility in the best scenario, the reverse is not true. However, for the sake of trust and advancement of Science, reproducibility is in need of a correctness verification mechanism. The following section details what this is currently.

7.3 INDEPENDENT VERIFICATION

In order to vet the correctness of a scientific result, other scientists must corroborate on its reproducibility and its veracity. To avoid any fraudulent activities in support of fake verifications, the scientists assessing the veracity of a result may do so independently. This process is called independent verification. This process has been extensively used in Quality Insurance and corporate review specially in accounting. In recent year, due to the growing number of frauds in research publications, independent verifications have been required in pharmaceutical, life Science and publication review committees. However, there is typically a cost incurred in implementing this process. Although possible as an extended feature of CoRR, independent verification and the complexity of managing it through a Web platform has not be investigated.

One might quickly state that with independent verification implemented, the reproducibility pipeline loop is guaranteed and close. However, there is still a situation not accounted for that points otherwise.

7.4 UNCONTROLLED RUNTIME VARIATIONS

When designing and properly scoping any system, accounting for failures is required. All systems eventually fail, while some are controllable, others are not. Furthermore, while some can be recovered from, others cannot. In the case of uncontrolled variations during the runtime of a computation, the cumulating impact can lead to a reproducibility issue. Situations such as

memory failure, data corruption, power surge and computational glitch are few cases for which reproducibility can be seriously impeded. As of today, there is no remedy to uncontrolled runtime variations and their potential impact of scientific results reproducibility. Worse, in some cases, results may be only able to be reproduced on one system due to sustained uncontrolled variations that are by essence non-transferable to another system as they just happen for various reasons.

While this is still an open problem that imped the guarantee of reproducibility for all current tools and platforms, there is another case in which all current solutions, even those presented in this thesis, will become unstable: scaling up. The next subsection briefly details this situation.

7.5 REPRODUCIBILITY AT EXASCALE

The major applications of Exascale computing are extremely large and complex simulations. Examples are: combustion, climate modeling, astrophysics, Aerospace, Airframes, Jet Turbines, etc… These simulations are typically the orchestration of millions or even billions of small computations. Ensuring the reproducibility of such big simulations is already a challenge on its own. In fact, the execution of all computations must be captured and stored. Thus, while storage is likely to be a problem, reproducing such a simulation requires special hardware (an Exascale machine). Besides these challenges, there are issues that while rare on regular computers, become very frequent at Exascale. Examples of those have been listed in the previous subsection. In essence, if a failure, like a soft error due to an alpha particle, occurs once in a trillion cycles for a Gigascale computer, at Exascale one failure will occur at every thousand cycles. Adding this to the storage and hardware challenge cause most existing solutions to become unstable or unpractical at this scale.

7.6 CONCLUSION

We have taken the reader here, into the realm of the confusion, the inadequate and the unpractical for this research solutions. In fact, the research conducted during this research tackle fundamental problems frequently faced by todays Scientists. As such, the situations listed here are beyond this scope by nonetheless critical. Correctness despite being different is in some cases

required to confirm reproducibility. However, this can't generally be done without independent validators. By not focusing in capitalizing on the latter specifically, these two situations present a limitation of our current solutions. From the four listed situations, the two first recalled previously can easily be dealt with through a new feature in CoRR. The other two cases are for now considered unsolved even if they occur in rare cases for one and for the other considered a problem of the few since Exascale machines are not to be found on every street corner.

Chapter 8 – GENERAL CONCLUSION

8.1 SUMMARY OF THE THESIS

This thesis takes on the core problem of the current reproducible research tools, platforms and approaches. Our work revolves around the research of an adequate representation of software execution context to ensure reproducibility. However, as shown in the literature review, the current spectrum of tools and platforms uncovers a wide range of representation approaches and variations. Thus, instead of reinventing the wheel and leading to yet another tool or platform, a deeper look and questioning was inevitable. While the actual questioning and motivation might seem subjective and may entice the reader for others, the actual benefit of such a research exercise is the extraction of problems of interests and motivations behind them that hopefully emerge from questions beyond our own. Therefore, our inquiry yielded four fundamental problems that must be addressed in diapason with the execution context representation. First, most tools and web platforms in support of reproducible research lack adequate means to reach out to scientists. Fundamentally, most do cope well with the importance of the World Wide Web in gathering the masses. Second, we are not aware of any reproducible support tool or platform that is interoperable with any of the other existing ones. That is, a record from one is equivalent to one of the other in a sense that it can be converted or directly ingested as is. Third, if the advent of the Internet, the Web and their various applications have taught us one thing is that collaboration in Science can be and must be boosted by any means possible. In fact, research communities are growing, new processes and methods are invented to be more advanced and to require new collaboration paradigms that old notions can't handle. This is the case in general and thus also for reproducible research support tools. Collaborative features are either partially proposed or not at all. Fourth and last, up to this moment of the thesis, we are also not aware of any approach that could consistently and systematically guarantee the exact numerical repeatability of research results. The motivations behind these problems were principally that the first three were correlated in a way that a viable solution could only exist if it solves the three altogether. Thus, the last problem's solution can be added to the one of the latter three to obtain a complete solution to our investigation and goals.

The first stage of this work presented in the previous paragraph is developed in chapter 2 and chapter 3. Although presented in reverse order to the presupposed dependency of one to the other, we want to stress to the reader that in practice, those two chapters' facts were interchangeably developed reciprocally.

In the second stage of this work, we research and develop the motivated solutions to our problematic. As a result, most of the time spent in this thesis was on the solution to the first three problems: the CoRR or Cloud of Reproducible Records. It's a web platform designed and constructed to reach the goal of pushing scientific awareness and collaboration further than ever before and provides a novel of offering interoperability to existing tools and platforms through its internal flexible execution context representation. Therefore, these platforms need not to worry but to keep up with the changes in CoRR only in the cases where the interactions protocol has drastically changed. We have designed the platform to avoid such cases and mainly focus on smooth updates with low consequences to backward compatibility. Also, by being a computational solution, CoRR interface can be adapted to tools and web platforms. Moreover, the software controlling experimental machines can be modified as done with tools and web platforms to communicate with an instance of CoRR. Thus, we have come to the conclusion that integrating a tool to CoRR to ensure the reproducibility of its managed computation is at a level of flexibility that its agnostic to whether the end unit is a standard computer or more of a complex custom machine.

The rest of the time spent in this thesis was on the solution to exact numerical results repeatability: Computation numerical results caching with Num-Cache. It's a software package that can be imported in scientific software to produce a cache of all intermediary computations and is sufficient to guarantee the whole computation's exact numerical repeatability.

Despite the adequacy of the thesis' results as novel solutions to the problems of interest, they present some limitations. In fact, in the case of uncontrolled runtime variations and at Exascale, they struggle where all attempts at these edges fail. Clearly CoRR and Num-Cache are ill-equiped against unexpected failures that beyond their features may render the latter inaccessible. By causing total or partial system failures, these random hard crashes jeopardize all executing software and stored data. Therefore, it is still a persistent issue. Already hard to manage on a

single Gigascale machine, uncontrolled runtime error and their twin pair controlled become a problem on Exascale machines. The problem is that there are just too many things going on that the controlled nature of the error is waived. They all occur very much more often. Moreover, we have deemed critical to open the discussion and clear a possible confusion between CoRR and Num-Cache being respectively a reproducibility booster and guarantor. Correctness requires advanced techniques of assessing veracity that is approached by involving independent verifications. It is getting more interest from communities of scientists across various fields as the ultimate assessment mechanism for trust.

Despite these limitations, the solutions of this thesis are proving useful in existing projects as the next subsection will present in addition to briefly reviewing CoRR and Num-Cache. Moreover, the perspective development of CoRR and Num-Cache allow a wide range of reflection and research to address their current limitations.

8.2 CONTRIBUTIONS OF THE THESIS

The vast majority of research activities carried out during this thesis took place at NIST within the Materials Measurements Laboratory. Naturally, a fair amount of our scientific contributions during this journey are clustered within the domain of Material Science. The contributions can be organized in two subsequent parts. The first part contains the two direct contributions that motivated this study and aimed at solving the selected problems stated in chapter 2. The second part regroups the indirect contributions which are mainly composed of collaborative work involving consulting invitation on the subject of reproducibility. Additionally, the latter part contains the applicative usages of one of the two direct contributions.

Con1. CoRR - The Cloud of Reproducible Records:

As introduced in our motivations O1 regarding problem statements $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$ we designed and implemented CoRR (Chapter 2). It aims at delivering 4 major features. First, it provides a generic representation model that can store any existing tool representation of what a reproducible artifact is. Thus, CoRR can serve as an interoperability bridge between the support tools. With its generic model, artifacts can freely transition from one tool to the other. Second, it currently integrates three major tools (Sumatra, ReproZip and CDE). As of version 0.1, users of

these tools can store their records on any instance of CoRR. The effort to integrate more tools is incremental and we hope to integrate the 40 software and services supporting reproducibility. Third, CoRR is a social network. It allows scientists to share records and collaborate around them by communicating their corroboration attempts outcomes. In addition, the fundamental web aspect of it makes the perfect framework for reaching out and driving awareness to the issues of reproducibility and the current solutions. The fourth and the last major feature is its federation capability. In fact, CoRR is designed to allow the run of multiple instances across various locations. Many institutions having their own constraints, it makes it ideal for internal usage, isolated from others. Yet, when needed, instances can be connected to each other. Hence, the search and collaborative capability of one instance is expended to all its connected pairs. With these features, we have reasons to believe that CoRR can substantially inhibit P1, P2 and P3. More features and details are provided in the upcoming chapters.

Con2. Computation Operations Caching for Numerical Repeatability:

By wrapping core operators (+, -, *, /), we can extract the order at which their operations are being evaluated. Moreover, by recording the two operands, the operator and the result as the operations unfold, we can generate a numerical operations cache. Then, fast retrieval hash-keys are produced by hashing the operands values with the operator in the order of the operation evaluation. This entire mechanism is implemented in the Num-Cache library. When the operations order is changed, Num-Cache can detect it through the generation of new hash with respect to the previous state of the cache. Thus, Num-Cache can alert scientists of an out of order mechanism that could cause precision issues. Also, when the order is preserved but the operation fails to deliver the same result, Num-Cache can reload the previous result and comparatively also throw an alert. With these two features, the scientist can furthermore task the library to adopt a new result or reload a previously cached one in the advent of an alert.

With the previous summary description of the novelty in Num-Cache, we are confident that it opens a new way to solve the problems of numerical precision due to our current processors' limitations. In fact, we dare hope that it will rise attention to processors manufacturer and spike

interests in the release of low-level code that will carry on with our research with Num-Cache and make processors smarter when computing operations involving big and small Floating-Point numbers. The solution here is to prioritize operations between numbers at same order of magnitude.

Con3. Evaluation and comparison of classical interatomic potentials through a user-friendly interactive web-interface:

This collaborative contribution is among the earliest applicative use cases of CoRR. Classical empirical potentials/force-fields provides atomistic insights into material phenomena through molecular dynamics and Monte Carlo simulations. Despite their wide applicability, a systematic evaluation of materials properties using such potentials and, especially, an easy-to-use user-interface for their comparison is still lacking. To address this deficiency, CoRR was used to record reproducible artifacts of computed energetics and elastic properties of variety of materials such as metals and ceramics using a wide range of empirical potentials and compared them to DFT as well as to experimental data, where available. The database currently consists of 3248 entries including energetics and elastic property calculations, and it is still increasing. We also include computational tools for convex-hull plots for DFT and FF calculations. The data covers 1471 materials and 116 force-fields.

Con4. Role of e-Collaborations in Scaling-Up Materials Innovation:

The present intellectual contribution occurred in the form of a rich aggregation of ideas, experiences and expertise in the aspects of e-Collaborations importance with regard to advancing Materials Science. Our contribution was most pressing on the aspects of innovation activities involving the sharing of data/knowledge, collaborative tools, workflow capture and management tools. In fact, the goal of the recently announced materials innovation initiatives such as the MGI is to substantially reduce the time and cost of materials design and deployment. Achieving this goal requires taking advantage of recent advances in data and information Sciences and fostering variety of existing and emerging modes of online collaborations between diverse stakeholders, hereafter collectively referred to as e-collaborations. These e-collaborations must become a core strategy to accomplish the vision of scaled-up materials innovation. Key ingredients needed for

successful adoption of e-collaborations in materials innovation activities include shareable and accessible data repositories, teaming tools, workflow capture and management tools, and annotation tools. The resulting paper reviews opportunities for scaled-up materials innovation through adoption of these emerging toolsets and presents a specific case study in modeling and design of Ni-based super alloys.

Con5. Data curation software for reproducible results in the FACT Lab:

This contribution has involved the supervision and training of a summer student for a trimester on the general subject of software engineering and more specifically on the aspect of reproducible research. It is the main experimentally focused exploration done during this thesis on the peculiar issues of reproducibility with laboratory machines and the research of solutions to remediate these. The goals of the MGI to reduce the time and cost of the deployment of new materials by 50% have increased the need to share experimental and computational data. Concurrent with the increased need to share data, the materials Science community has increased the emphasis on the data reproducibility as well. To enable the ability to share data and provide reproducible data, a variety of data curation tools are being developed. One of these curation tools is the NIST Information Technology Laboratory, Materials Data Curation System. This data curation tool is a Python/MongoDB/Django Web-based system that provides a means for capturing, sharing, and transforming data. This is being used to develop a data curation protocol for high pressure CO2 adsorption isotherms measured at 20 °C on the NIST reference material RM-8852 (zeolite ZSM-5) using state-of-the-art instruments in the FACT Lab, a laboratory commissioned to establish testing procedures and provide reliable material property data. We aimed to investigate and build a validation workflow to address the effectiveness of the testing procedures and the reliability of the material property data. The data is converted directly from Excel into a JSON and XML format using a python script. The XML formatted data can then be entered into the curator using an API. Once the data are entered into the MDCS, the data can be searched and shared with other users. The result of this workflow will help refine standards of how the experimental results are gathered and curated to the MDCS, while also improving

how other scientists should use these results in the best way to improve reproducibility across further computations.

Con6. CHiMAD Benchmark computations in CoRR:

This contribution has involved the supervision and training of a second summer student for a trimester on the aspect of using the early integration of Sumatra with CoRR to record computations for a Benchmark. CHiMaD is a phase field community dedicated to distributing phase field models to determine the most efficient way of simulating various materials. The goal of this contribution was to record the computations and evaluate the performance in terms of execution time, memory consumption, CPU usage and convergence speed of various phase field codes on 6 major phase field problems: Spinodal Decomposition, Ostwald Ripening, Dendritic Growth, Linear Elasticity, Stokes Flow, Electrostatics and MMS Allen-Cahn.

Con7. Open Interoperability Project between CoRR and Tools:

This contribution derives from a collaboration with the MaestroWF team from LLNL. Before this collaboration, integrations of tools to CoRR were done within each tool source code. This obviously led to duplicated code that becomes hard to maintain as they evolve separately within each tool. The outcome of this collaboration is a python library named **contracts** that contains all the common mechanisms of interacting with the CoRR API and proposes tool to tool communication with CoRR as a translating bridge. Thus, to integrate a new tool with CoRR now signifies integrating this library in the tool source code as done with other tools. Therefore, when the contracts library evolves, it does so for all the tools homogeneously.

8.3 PERSPECTIVES

The limitations to the solutions contributed can be addressed in two separate future timeframes. Among these limitations are those that can be addressed relatively easily through features added to CoRR and Num-Cache. However, there are others that will require more questioning and research. The latter may well possibly open the way to new directions and complementary approaches.

8.3.1 Future directions with CoRR

The collaboration schemes provided by CoRR already include the possibility for scientists to link two records by stating if they are repeats, reproductions or replicates. Although, this still does not mean that CoRR goes beyond reproducibility to correctness, we are confident that the same mechanism can be used to allow independent verifications. Thus, scientists can be invited to be independent verifiers and will be able to assess the veracity of records as currently done with scientific papers reviews invitations.

CoRR is a web platform and by design is not supposed to cohabitate with the tools and web platforms that it integrates. The latter runs inside computing environments or external cloud platforms. Thus, in the case of a runtime error on the system in which the traced computation is executing, CoRR is able to determine the nature of the failure based on previous runs. Such errors include incomplete data reception, data corruption, dependency change, source code alteration, etc. However, CoRR is not designed to handle uncontrolled runtime error that leads to total system failures on its own computing node or the traced computations. The only evidence will be not updated status still at running while the link to the process vanished abruptly in an uncontrolled fashion. The problem of scale for CoRR resides at two points. First, the tool watching the computations on the Exascale infrastructure must be able to watch the potential billions of threads running from a single computation launch. This is where the initial challenge starts. Second, as any modern web platform/application that serves users by their millions, a CoRR instance that serves scientists across an Exascale machine needs an appropriate cloud infrastructure with the proper load balancing and high availability.

8.3.2 The possibilities ahead of Num-Cache

By leveraging a caching approach that stores the mathematical operations of computations in association with their results in the order of their evaluation, Num-Cache opens the way for new techniques in numerical results repeatability. However, at the moment, the main issue of its very first version is the additional time incurred in every operation secured by the library. We have theoretically evaluated it to be in the order of O(log n). The main challenge going forward is the application of techniques to reduce this time lost at best. Indeed, this will move Num-Cache from

its suitability at debug time into its adequacy de facto at every stage of the software lifecycle. Interesting and insightful comments and recommendations from participants at SummerSim edition 2018 suggest three main options going forward. First, instead of a ad hoc library, inject some code from the compiler in the case of C++ or the interpreter in the case of Python. Second, a study on the impact on the hardware used may reveal being useful. Indeed, the cost of O(log n) may just reveal itself to be more acceptable in the case of a GPU based computation than a CPU one. Third, maybe we should resign ourselves at advertising this method only at debug time. Depending on the second option, this may well be limited to CPU and not GPU based computation. Last and not the least, in our opinion, we think Num-Cache can be more effective with a more adaptable and more secure core. We think of more dynamic features involving interruptions and requesting the user to use the strategy at the moment for the specific operation or for all operations. Additionally, we plan to integrate a RSA public encryption to protect the cache from unwanted eyes should the scientists feel skeptical in dumping their results in clear for all to see.

8.3.3 Ongoing and Future directions in General

Going forward with the results of this thesis involves three critical activities. The first activity is more personal and involves pushing the approaches and techniques used in these results further. We look into bettering each tool but furthermore into fully integrating CoRR and Num-Cache. Including future plans for each result separately, we also look forward into other aspect of scientific computing that are complementary to reproducibility. The aspect of correctness and specifically in ways to automate independent verifications is of interest to us. In fact, it has already received a certain level of thoughts in which we see blockchain as the modern-day technology that can make such a solution possible and viable. The second activity involves collaborations with tools developers. We are looking into yearly meeting in which we could drive the various disparities into common understanding in which CoRR can play a key role. We have to come to an agreement that despite the common goal and the overlaps most of these tools are unique as they mostly aim at solving the problem from a different point of view. The latter is strongly motivated by the specifics of the domain. A neuroscientist indeed may not see things the same way as a geologist specially when designing a software. Moreover, if we cannot find

reasons for differences may be various teams could merge efforts and thus move quicker to what they are hoping to achieve. Therefore, such an activity will involve discussions into representations mapping between tools through CoRR. This will be fueled by our proposals. Additionally, we will have a common ground understanding on things in which no team can compromise on and build groups of tools per similarities. Teams could then work together in learning from each other after presentations from each team. The third activity involves hands on hackathons in which tools developers can show scientists how to use their tools with CoRR. More generally we are looking to make our mission to democratize tools and platforms in service of improving reproducibility to all scientists at large. All our activities will be set at an International scale and thus be open to anyone through public publications sources for our content and streaming services for our meetings.

References

Afgan, E., Baker, D., van den Beek, M., Blankenberg, D., Bouvier, D., Cech, M., Chilton, J., et al. (2016). The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic Acids Research*, 44(W1):W3–W10. doi:10.1093/nar/gkw343. (Cited on page 79.)

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., and Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 443–424, Santorini Island, Greece. doi:10.1109/SSDM.2004.1311241. (Cited on pages 44 and 52.)

Amstutz, P., Crusoe, M. R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D. et al. (2016). Common Workflow Language v1.0. *Common Workflow Language Working Group*, doi:10.6084/m9.figshare.3115156.v2. (Cited on page 54.)

Andel, C., Davidow, S. L., Hollander, M., and Moreno, D.A. (2012). The economics of health care quality and medical errors. *Journal of Health Care Finance,* 39(1):39–50. (Cited on page 20.)

Andersen, H., and Hepburn, B. (2018). Scientific Method. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University. (Cited on page 27.)

Aristotle, Cooke, H. P., Tredennick, H., and Forster, E. S. (1938). *The Organon*. Cambridge, Harvard University Press. (Cited on page 27.)

Badia, R. M., Ayguade, E., and Labarta, J. (2017). Workflows for Science: A Challenge when Facing the Convergence of HPC and Big Data. *Supercomputing Frontiers and Innovations*, volume 4, issue 1, pages 27–47. (Cited on page 44.)

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, volume 533, pages 452–454. (Cited on page 20.)

Barba, L. A. (2018). Terminologies for reproducible research. *Arxiv preprint arXiv:1802.03311*. (Cited on page 85.)

Barth, T. (2011). A Brief Overview of Uncertainty Quantification and Error Estimation in Numerical Simulation. *NASA Ames Research Center*. Lecture. (Cited on page 68.)

Bartlett, J. W. and Frost, C. (2008). Reliability, repeatability and reproducibility: analysis of measurement errors in continuous variables. *Ultrasound in Obstetrics & Gynecology Reliability*, 31(4):466–75. doi:10.1002/uog.5256. (Cited on page 86.)

Bell, G. (2016). Replicates and repeats. *BMC Biology*, volume 14, pages 28–29, doi:10.1186/s12915-016-0254-5. (Cited on page 85.)

Bellard, F. (2005). QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track,* pages 41–46. (Cited on page 63.)

Best, M. L. (2004). Can the Internet be a Human Right? *Human Rights & Human Welfare*, volume 4, issue 1, pages 23–31. (Cited on page 72.)

Bhardwaj, V. (2015). Villain of Molecular Biology: Why are we not reproducible in research? *F1000Research 2015*, 4:438. doi:10.12688/f1000research.6854.1. (Cited on page 26.)

Bird, A. 2008. Scientific progress as accumulation of knowledge: a reply to Rowbottom. *Studies in History and Philosophy of Science Part A*, volume 39, issue 2, pages 279–281. doi:10.1016/j.shpsa.2008.03.019. (Cited on page 27.)

Bohlender, G., Cordes, D., Knofel, A., Kulisch, U., Lohner, R., and Walter, W. V. (1993). Proposal for Accurate Floating-Point Vector Arithmetic. *Mathematics in Science and Engineering*, volume 189, pages 87–102. doi:10.1016/S0076-5392(08)62843-X. (Cited on page 66.)

Bucher, C. G., and Bourgund, U. (1990). A fast and efficient response surface approach for structural reliability problems. *Structural Safety*, volume 7, issue 1, pages 57–66, doi:10.1016/0167-4730(90)90012-E. (Cited on page 67.)

Burke, R. B. (1962). *The Opus Majus Of Roger Bacon Volume II*. Russell & Russell Inc. (Cited on page 21.)

Callahan, S., Freire, J., Santos, E., Scheidegger, C., Silva, C. and Vo, H. (2006). VisTrails: Visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international*

conference on Management of data, pages 745–747, Chicago, IL, USA. ACM Press. doi:10.1145/1142473.1142574. (Cited on page 44.)

Chirigati, F., Rampin, R., Shasha, D., and Freire, J. (2016). Reprozip: Computational reproducibility with ease. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2085–2088, San Francisco, CA, USA. ACM Press. doi:10.1145/2882903.2899401. (Cited on pages 34 and 44.)

Dao, V. T., Maigne, L., Breton, V., Nguyen, H. Q., and Hill, D. R. C. (2014). Numerical Reproducibility, Portability and Performance of Modern Pseudo Random Number Generators: Preliminary study for parallel stochastic simulations using hybrid Xeon Phi computing processors. *Modelling and Simulation 2014 - European Simulation and Modelling Conference, ESM 2014*, pages 80–87, Porto, Portugal. (Cited on pages 142.)

Davison, A. (2014). Sumatra: A Toolkit for Reproducible Research. In Stodden, V., Leisch, F., and Peng, R. D., editors, *Implementing Reproducible Research*, pages 57–79. Chapman and Hall/CRC. (Cited on pages 34 and 44.)

Davison, A. (2016). Reproducibility through environment capture: Part 1: Docker. *HBP CodeJam Workshop #7*, https://goo.gl/9SKN7y. (Cited on pages 56, 58 and 61.)

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, volume 51, issue 1, pages 107–113. doi:10.1145/1327452.1327492. (Cited on page 38.)

Demartini, G., and Siersdorfer, S. (2010). Dear search engine: What's your opinion about…?: Sentiment analysis for semantic enrichment of web search results. In *Proceedings of the 3rd International Semantic Search Workshop*, article 4, Raleigh, NC, USA. doi:10.1145/1863879.1863883. (Cited on page 33.)

den Exter, K., Rowe, S., Boyd, W., and Lloyd, D. 2012. Using Web 2.0 Technologies for Collaborative Learning in Distance Education—Case Studies from an Australian University. *Future Internet*, 4(1):216-237. doi:10.3390/fi4010216. (Cited on page 75.)

Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., and Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature Biotechnology*, volume 35, issue 4, pages 316–319, doi:10.1038/nbt.3820. (Cited on page 44.)

Dolan-Gavitt, B., Hodosh, J., Hulin, P., Leek, T., and Whelan, R. (2015). Repeatable reverse engineering with PANDA. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, article 4, Los Angeles, CA, USA. doi:10.1145/2843859.2843867. (Cited on pages 44 and 63.)

Donoho, D. L. (2010). An invitation to reproducible computational research. *Biostatistics,* volume 11, issue 3, pages 385–388. doi:10.1093/biostatistics/kxq028. (Cited on page 42.)

Drummond, C. (2009). Replicability is not Reproducibility: Nor is it Good Science. *The 4th workshop on Evaluation Methods for Machine Learning*, Montreal, Canada. (Cited on page 127.)

Duvendack, M., Palmer-Jones, R., and Reed, W. R. (2017). What is meant by "replication" and why does it encounter resistance in economics? *American Economic Review*, volume 107, issue 5, pages 46–51. doi:10.1257/aer.p20171031. (Cited on page 86.)

Eisner, D. A. (2018). Reproducibility of Science: Fraud, impact factors and carelessness. *The Journal of Molecular and Cellular Cardiology*, 114:364–368. doi:10.1016/j.yjmcc.2017.10.009. (Cited on page 20.)

Folk M., Heber, G., Koziol, Q., Pourmal, E., and Robinson, D. (2011). An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47, Uppsala, Sweden. ACM Press. doi:10.1145/1966895.1966900. (Cited on page 48.)

Foster, E., and Deardorff, A. (2017). Open Science Framework (OSF). *The Journal of the Medical Library Association*, 105(2): 203–206. doi:10.5195/jmla.2017.88. (Cited on page 80.)

Franzen, M. (2016). Science Between Trust and Control: Non-Reproducibility in Scholarly Publishing. In Atmanspacher, H. and Maasen, S., editors, *Reproducibility: Principles, problems, practices and prospects*, chapter 22, pages 468–485, Wiley. doi:10.1002/9781118865064.ch22. (Cited on page 26.)

Gantikow, H. (2017). Linux Containers - Technology and Runtimes. *ISC2017 Workshop: Linux Containers to Optimise IT Infrastructure for HPC & BigData.* https://goo.gl/ARYyU9. (Cited on pages 44, 59 and 61.)

Gavish, M., and Donoho, D. L. (2011). A universal identifier for computational results. In Sato, M., Matsuoka, S., Sloot, P. M., van Albada, G. D., and Dongarra, J., editors, *Proceedings of the International Conference on Computational Science, ICCS 2011,* volume 4, pages 637–647. Elsevier. doi:10.1016/j.procs.2011.04.067. (Cited on page 44.)

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *The ACM Computing Surveys (CSUR)*, volume 23, issue 1, pages 5–48. ACM Press. doi:10.1145/103162.103163. (Cited on pages 38 and 129.)

Gordon-McKeon, S. (2015). What we talk about when we talk about replication. *Open Science Collaboration*. http://osc.centerforopenscience.org/2014/08/07/talk-about-replication/. (Cited on page 127.)

Gorelik, E. (2013). Cloud Computing Models. *Massachusetts Institute of Technology*, http://web.mit.edu/smadnick/www/wp/2013-01.pdf. (Cited on page 74.)

Gorgolewski, K., Burns, C. D., Madison, C., Clark, D., Halchenko, Y. O., Waskom, M. L., and Ghosh, S. S. (2011). Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in Python. *Frontiers in Neuroinformatics* vol. 5 article 13, 15 p. doi:10.3389/fninf.2011.00013. (Cited on page 44.)

Gorp, P. V., and Mazanek, S. (2011). Share: a web portal for creating and sharing executable research papers. *International Conference on Computational Science: Procedia Computer Science* vol. 4, pp. 589–597. doi:10.1016/j.procs.2011.04.062. (Cited on page 44.)

Guo, P. (2012). CDE: A Tool for Creating Portable Experimental Software Packages. *Computing in Science & Engineering*, volume 14, issue 4, pages 32–35. doi:10.1109/MCSE.2012.36. (Cited on pages 32 and 44.)

Guthrie, S., Connelly, A., Amstutz, P., Berrey, A. F., Cesar, N., Chen, J., Chippada, Ret al. (2015). Tiling the genome into consistently named subsequences enables precision medicine and

machine learning with millions of complex individual data-sets. *PeerJ PrePrints,* volume 3, article e1426v1. doi:10.7287/peerj.preprints.1426v1. (Cited on page 44.)

Heimlich, O. (2015). Interval arithmetic in GNU Octave, SWIM 2016. https://swim2016.sciencesconf.org/data/pages/Heimlich.pdf. (Cited on page 62.)

Hilbert, M., and Lopez, P. (2011). The world's technological capacity to store, communicate and compute information. *Science,* volume 332, issue 6025, pages 60–65. doi: 10.1126/science.1200970. (Cited on page 91.)

Hill, D. R. C. (2017). Numerical reproducibility of Parallel and Distributed Stochastic Simulations using High Performance Computing. In Traore, M. K., editor*, Computational Frameworks: Systems, Models and Applications*, chapter 4, pages 95–109. Eslevier. (Cited on page 128.)

Hill, D. R. C. (2015). Parallel Random Numbers, Simulation, Science and reproducibility. *Computing in Science & Engineering*, volume 17, issue 4, pages 66–71. (Cited on pages 65 and 67.)

Hill, D. R. C., Passerat-Palmbach, J., Mazel, C., and Traore, M. K. (2013). Distribution of Random Streams for Simulation Practitioners. *Concurrency and Computation: Practice and Experience,* volume 25, issue 10, pages 1427–1442. (Cited on page 67.)

Hinsen, K. (2015). ActivePapers: a platform for publishing and archiving computer-aided research. F1000Research 2015, 3:289. (Cited on page 44.)

Hothorn, T., and Leisch, F. (2011). Case studies in reproducibility. *Briefings in Bioinformatics*, 12(3):288–300. doi:10.1093/bib/bbq084. (Cited on page 43.)

Introna, L., and Nissenbaum, H. (2000). Shaping the Web: Why the Politics of Search Engines Matters. *Information Society*, 16(3): 169–185. (Cited on page 72.)

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., et al. (2015). FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, volume 27, issue 17, pages 5037–5059. (Cited on page 44.)

JCGM. (2008). The international vocabulary of metrology—basic and general concepts and associated terms (VIM). *Joint Committee for Guides in Metrology, 3rd edition*. (Cited on page 86.)

Kerber, W., and Schweitzer, H. (2017). Interoperability in the Digital Economy. *The Journal of Intellectual Property, Information Technology and Electronic Commerce Law*, 8(1): 39 –58. (Cited on page 28.)

Kling, R. (1991). Computerization and Social Transformations. *Science, Technology, & Human Values*, 16(3): 342–367. doi:10.1177/016224399101600304. (Cited on page 72.)

Knuth, D. E. (1984). Literate Programming. *The Computer Journal*, 27(2): 97–111. (Cited on page 44.)

Knuth, D. E. (1992). Literate Programming. Center for the Study of Language and Information. ISBN 0-937073-80-6. (Cited on page 45.)

Koc, A., and Tansel, A. U. (2011). A survey of version control systems. In *The 2nd International Conference on Engineering and Meta-Engineering: ICEME 2011*, Orlando, FL, USA. (Cited on page 32.)

Kraker, P., Lex, E., Gorraiz, J., Gumpenberger, C., and Peters, I. (2015). Research data explored II: The anatomy and reception of figshare. In *Books of Abstracts of Research organizations under scrutiny: New indicators and analytical results. 20th International Conference on Science and Technology Indicators*, pages 77–79, Lugano, Switzerland. (Cited on page 48.)

Kurt, L. E., and DeSalle, R. (2009). Evidence, Content and Corroboration and the Tree of Life. *Acta Biotheoretica*, 57(1–2):187–199. Springer Netherlands. (Cited on page 27.)

Kurzweil, R. (2004). The Law of Accelerating Returns. In Teuscher, C., editor, *Alan Turing: Life and Legacy of a Great Thinker*, pages 381–416. Springer Berlin Heidelberg. (Cited on page 21.)

LeVeque, R. J., Mitchell, I. M., and Stodden, V. (2012). Reproducible research for scientific computing: tools and strategies for changing the culture. *Computing in Science & Engineering*, volume 14, pages 13–17. (Cited on page 31.)

Lomas, C., Burke, M., and Page, C. (2008). Collaboration Tools. Educause Learning Initiative: Advancing Learning Through IT Innovation, ELI, paper 2. (Cited on page 75.)

McNutt, M. (2014). Journals unite for reproducibility. *Science*, volume 346, issue 6210, page 679. (Cited on page 26.)

Morabito, R., Kjallman, J., and Komu, M. (2015). Hypervisors vs. lightweight virtualization: a performance comparison. *2015 IEEE International Conference on Cloud Engineering*, pages 386–393, Tempe, AZ, USA. doi:10.1109/IC2E.2015.74.(Cited on page 58.)

Munafò, M. R., Nosek, B. A., Bishop, D. V. M., Button, K. S., Chambers, C. D., Percie du Sert, N., Simonsohn, U. et al., (2017). A manifesto for reproducible science. *Nature Human Behaviour*, volume 1, article 0021. (Cited on page 31.)

Nehmeier, M. (2014). libieeep1788: A C++ Implementation of the IEEE interval standard P1788. *2014 IEEE Conference on Norbert Wiener in the 21st Century (21CW)*, pages 1–6, Boston, MA, USA. doi:10.1109/NORBERT.2014.6893854. (Cited on page 66.)

Nishimura, T., and Matsumoto, M. 1998. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation*, 8(1):3–30. (Cited on page 67.)

NRE. (2015). Numerical Reproducibility at Exascale. https://goo.gl/iqMZw7. (Cited on page 142.)

Oinn, T., Addis, M., Ferris, J., Marvin, D., Greenwood, M., Carver, T., Pocock, M. R., Wipat, A., and Li, P. (2004). Taverna: a tool for Creating Portable Experimental Software Packages. *Bioinformatics*, 20(17):3045–3054. doi:10.1093/bioinformatics/bth361. (Cited on page 80.)

Orvis, J., Crabtree, J., Galens, K., Gussman, A., Inman, J. M., Lee, E., Nampally, S. et al., (2010). Ergatis: a web interface and scalable software system for bioinformatics workflows. *Bioinformatics*, 26(12):1488–1492. (Cited on page 79.)

Ovadia, S. (2014). Markdown for librarians and academics. *Behavioral & Social Sciences Librarian*, 33(2):120–124. doi:10.1080/01639269.2014.904696. (Cited on page 52.)

Pautasso, C., Zimmermann, O., and Leymann, F. (2018). Restful web services vs. “big'” web services: making the right architectural decision. In *WWW '08 Proceedings of the 17th*

international conference on World Wide Web, pages 805–814, Beijing, China. ACM Press. doi:10.1145/1367497.1367606. (Cited on page 49.)

Perez, F., and Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, volume 9, issue 3, pages 21–29. (Cited on page 44.)

Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. (2017). noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts. In *Proceedings of VLDB Endowment*, volume 10, issue 12, pages 1841–1844. doi:10.14778/3137765.3137789. (Cited on page 44.)

Ratliff, J. D., and Rubinfeld, D. L. (2010). Online Advertising: Defining Relevant Markets. *Journal of Competition Law and Economics*, 6(3):653–686. doi:10.1093/joclec/nhq011. (Cited on page 72.)

Raychaudhuri, S. (2008). Introduction to Monte Carlo simulation. In *WSC '08 Proceedings of the 40th Winter Simulation Conference*, pages 91–100, Miami, FL, USA. Winter Simulation Conference. (Cited on page 66.)

Reuillon, R., M. Leclaire, and S. Rey-Coyrehourcq. (2013). OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Generation Computer Systems*, 29(8):1981–1990. Elsevier. doi:10.1016/j.future.2013.05.003. (Cited on page 44.)

Revol, N., and Theveny, P. (2014). Numerical Reproducibility and Parallel Computations: Issues for Interval Algorithms. *IEEE Transactions on Computers*, 63(8):1915–1924. doi:10.1109/TC.2014.2322593. (Cited on page 66.)

Rocklin, M., Huff, K., and Bergstra, J. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th Python in Science Conference (Scipy 2015)*, pages 130–136, Austin, TX, USA. (Cited on pages 44 and 54.)

Rosebrock, A. (2018). *The Deep Learning for Computer Vision with Python*. PyImageSearch. (Cited on page 111.)

Rosenthal, D. S. H. (2010). Format obsolescence: assessing the threat and the defenses. *Library Hi Tech*, 28(2):195–210. doi:10.1108/07378831011047613. (Cited on page 45.)

Rosenthal, D. S. H. (2015). *Emulation & virtualization as preservation strategies*. UNT Digital Library. (Cited on page 63.)

Rubin, J. (2015). Federated Systems. *Massachusetts Institute of Technology*. Technical. (Cited on page 75.)

Rule, A., Tabard, A., and Hollan, J. D. (2018). Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*, paper 32, Montreal, QC, Canada. ACM Press. doi:10.1145/3173574.3173606. (Cited on page 51.)

Sandve, G.K., A. Nekrutenko, J. Taylor, and E. Hovig. (2013). Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*, 9(10): e1003285. doi:10.1371/journal.pcbi.1003285. (Cited on pages 41 and 128.)

Slezák, P., and Waczulíková, I. (2011). Reproducibility and Repeatability. *Physiological Research,* 60(1):203–204, author reply 204–205. (Cited on page 127.)

Smith, C. H. L. (1997). What's the use of basic Science? https://goo.gl/wjbbbf. (Cited on page 20.)

Snyder, D. M. (2000). A Connection between Gravitation and Electromagnetism. *Arxiv preprint physics/0002028*. (Cited on page 27.)

Somers, J. (2018). The Scientific Paper Is Obsolete. https://goo.gl/YtrEgj. (Cited on page 44.)

Srinivasan, R. (1995). RPC: Remote procedure call protocol specification version 2. *RFC Editor*. doi:10.17487/RFC1831. (Cited on page 49.)

Stephen, A. T. (2016). The role of digital and social media marketing in consumer behavior. *Current Opinion Psychology*, 10:17–21. doi:10.1016/j.copsyc.2015.10.016. (Cited on page 72.)

Stodden, V. (2017). Implementing Reproducible Computational Research: Policy and Practice. Presentation. *Census*. (Cited on page 20.)

Stodden, V., Leisch, F., and Peng, R. D. (2014). *Implementing Reproducible Research*. Chapman and Hall/CRC. ISBN 9781466561595. (Cited on page 41.)

Strijkers, R., Cushing, R., Vasyunin, D., de Laat, C., Belloum, A. S. Z., and Meijer, R. (2011). Toward executable scientific publications. In *Proceedings of the International Conference on Computational Science, ICCS 2011: Procedia Computer Science*, volume 4, pages 707–715. (Cited on page 44.)

Taylor, K. T., and Taylor, S. (2018). The Scientific Paper Is Not Obsolete… It's Inaccessible. Untold. https://goo.gl/DiVMWX. (Cited on page 44.)

Turnbull, J. (2014). *The Docker Book*. https://goo.gl/QH1S9f. (Cited on pages 44 and 58.)

White, A. (2015). Federal agencies announce materials data challenge. *MRS Bulletin*, 40(11):906–907. (Cited on page 79.)

Wood, J. (2017). All about rkt: Containers and Kubernetes at CoreOS. *CoreOS*. Presentation. https://goo.gl/GSuFPo. (Cited on pages 44 and 59.)

Zitzlsberger, G. (2014). FP Accuracy & Reproducibility: Intel® C++/Fortran Compiler, Intel® Math Kernel Library and Intel® Threading Building Blocks. *Optimization Notice, Intel*. Presentation. (Cited on pages 37, 44, 65 and 129.)

Bibliography

List of published papers

1. Choudhary, K., Congo, F. Y. P., Liang, T., Becker, C., Hennig, R. G., Tavazza, F. (2017). Evaluation and comparison of classical interatomic potentials through a user-friendly interactive web-interface. *Scientific Data*, 4:160125.
2. Congo, F. Y. P., Traoré, M. K., Hill, D. R. C. (2018). Computation Operations Caching for Numerical Repeatability. In *SummerSim '18 Proceedings of the 50th Computer Simulation Conference*, article 28, Bordeaux, France. Society for Computer Simulation International.
3. Hill, D. R. C, Congo, F. Y. P., Dao Van, T., Schweitzer, P. (2015). Numerical Reproducibility for Parallel Stochastic Simulation "Exascale Ready". In *Numerical Reproducibility Workshop (NRE 2015): Super Computing 2015*, Austin, Texas, USA. ACM.
4. Kalidindi, S. R., Brough, D., Li, S. Y., Cecen, A., Blekh, A. L., Congo, F. Y. P., Campbell, C. E. (2016). Role of materials data Science and informatics in accelerated materials innovation. *MRS Bulletin*, 41(8):596–602.

List of papers accepted for publication

1. Choudhary, K., Zhang, Q., Chowdhury, S., Nguyen, N., Trautt, Z., Newrock, M., Congo, F. Y. P., Reid, A., Tavazza, F. (2018). Towards Efficient Optoelectronic Material Design using Density Functional Theory. Experiments and Machine Learning, Electronic and Advanced Materials (American Ceramic Society). in press.
2. Gabriel, J. J., Congo, F. Y. P., Sinnott, A., Matthew, K., Allison, T. C., Tavazza, F. M., Hennig, R. G. (2018). Uncertainty Quantification for Materials Properties in Density Functional Theory with k-Point Density. Current Opinion in Solid State & Materials Science. in press.

List of submitted papers

1. Congo, F. Y. P., Traoré, M. K., Wheeler, D., and Hill, D. R. C. (2018). CoRR - The Cloud of Reproducible Records. submitted. under review.