Deep Software Variability and Frictionless Reproducibility

Mathieu Acher @acherm
Deep Software Variability and Frictionless Reproducibility

Abstract: The ability to recreate computational results with minimal effort and actionable metrics provides a solid foundation for scientific research and software development. When people can replicate an analysis at the touch of a button using open-source software, open data, and methods to assess and compare proposals, it significantly eases verification of results, engagement with a diverse range of contributors, and progress. However, we have yet to fully achieve this; there are still many sociotechnical frictions. Inspired by David Donoho's vision, this talk aims to revisit the three crucial pillars of frictionless reproducibility (data sharing, code sharing, and competitive challenges) with the perspective of deep software variability.

Our observation is that multiple layers — hardware, operating systems, third-party libraries, software versions, input data, compile-time options, and parameters — are subject to variability that exacerbates frictions but is also essential for achieving robust, generalizable results and fostering innovation. I will first review the literature, providing evidence of how the complex variability interactions across these layers affect qualitative and quantitative software properties, thereby complicating the reproduction and replication of scientific studies in various fields. I will then present some software engineering and AI techniques that can support the strategic exploration of variability spaces. These include the use of abstractions and models (e.g., feature models), sampling strategies (e.g., uniform, random), cost-effective measurements (e.g., incremental build of software configurations), and dimensionality reduction methods (e.g., transfer learning, feature selection, software debloating).

I will finally argue that deep variability is both the problem and solution of frictionless reproducibility, calling the software science community to develop new methods and tools to manage variability and foster reproducibility in software systems.

Exposé invité, 5 juin 2024 @ GDRGPL
Special thanks to* Aaron Randrianaina, Jean-Marc Jézéquel, Benoit Combemale, Luc Lesoil, Arnaud Gotlieb, Helge Spieker, Quentin Mazouni, Paul Temple, Gauthier Le Bartz Lyan, Xhevahire Tërnava, Olivier Barais, and the whole DiverSE and RIPOST teams

*random order, incomplete
Frictionless Reproducibility and (Deep) Software (Variability)

Problem: Variability and Frictions

Solution: Variability and Exploration

Discussions
SOFTWARE VARIANTS ARE EATING THE WORLD
Science is changing: Computation-based research
Computational science depends on software and its engineering

design of mathematical model
mining and analysis of data
executions of large simulations
problem solving
executable paper

from a set of scripts to automate the deployment to... a comprehensive system containing several features that help researchers exploring various hypotheses
Computational science depends on software and its engineering

multi-million line of code base
multi-dependencies
multi-systems
multi-layer
multi-version
multi-person
multi-variant

Dealing with software collapse: software stops working eventually
Konrad Hinsen 2019
Configuration failures represent one of the most common types of software failures Sayagh et al. TSE 2018
“Insanity is doing the same thing over and over again and expecting different results”

Reproducibility

“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”
(Claerbout/Donoho/Peng definition)

“The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.” (~executable paper)
Reproducibility and Replicability

**Reproducible**: Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.

**Replication**: A study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new analyses.

“Terminologies for Reproducible Research”, Lorena A. Barba, 2018
Reproducible: Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.

Replication: A study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new analyses.

“Terminologies for Reproducible Research”, Lorena A. Barba, 2018

The Claerbout/Donoho/Peng terminology is broadly disseminated across disciplines (see Table 2). But the recent adoption of an opposing terminology by two large professional groups—ACM and FASEB—make standardization awkward. The ACM publicizes its rationale for adoption as based on the International Vocabulary of Metrology, but a close reading of the sources makes this justification tenuous. The source of the FASEB adoption is unclear, but there’s a chance that Casadevall and Fang (2010) had an influence there. They, in turn, based their definitions on the emphatic but essentially flawed work of Drummond (2009).

Table 2: Grouping of terminologies, as in Table 1, but by discipline.

<table>
<thead>
<tr>
<th>A</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>political science</td>
<td>signal processing</td>
<td>microbiology, immunology</td>
</tr>
<tr>
<td>economics</td>
<td>scientific computing</td>
<td>(FASEB)</td>
</tr>
<tr>
<td></td>
<td>econometry</td>
<td>computer science (ACM)</td>
</tr>
<tr>
<td></td>
<td>epidemiology</td>
<td></td>
</tr>
<tr>
<td></td>
<td>clinical studies</td>
<td></td>
</tr>
<tr>
<td></td>
<td>internal medicine</td>
<td></td>
</tr>
<tr>
<td></td>
<td>physiology (neuro)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>computational biology</td>
<td></td>
</tr>
<tr>
<td></td>
<td>biomedical research</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statistics</td>
<td></td>
</tr>
</tbody>
</table>

*As a result of discussions with the National Information Standards Organization (NISO), it was recommended that ACM harmonize its terminology and definitions with those used in the broader scientific research community, and ACM agreed with NISO’s recommendation to swap the terms “reproducibility” and “replication” with the existing definitions used by ACM as part of its artifex review and badges initiative. ACM took action to update all prior badges to ensure consistency.*
Reproducibility and Replicability

**Methods Reproducibility:** A method is reproducible if reusing the original code leads to the same results.

**Results Reproducibility:** A result is reproducible if a reimplementation of the method generates statistically similar values.

**Inferential Reproducibility:** A finding or a conclusion is reproducible if one can draw it from a different experimental setup.

"Unreproducible Research is Reproducible", Bouthillier et al., ICML 2019
Reproducible science

“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”

Socio-technical issues: open science, open source software, multi-disciplinary collaboration, incentives/rewards, initiatives, etc. with many challenges related to data acquisition, knowledge organization/sharing, etc.
Reproducible science

“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”

Socio-technical issues: open science, open source software, multi-disciplinary collaboration, incentives/rewards, initiatives, etc. with many challenges related to data acquisition, knowledge organization/sharing, etc.

EMSE Open Science Initiative

Openness in science is key to fostering progress via transparency, reproducibility, and replicability. Especially open data and open source are two fundamental pillars in open science as both build the core for excellence in evidence-based research. The Empirical Software Engineering journal (EMSE) has therefore decided to explicitly foster open science and reproducible research by encouraging and supporting authors to share their (anonymised and curated) empirical data and source code in form of replication packages. The overall goals are:

- Increasing the transparency, reproducibility, and replicability of research endeavours. This supports the immediate credibility of authors’ work, and it also provides a common basis for joint community efforts grounded on shared data.
- Building up an overall body of knowledge in the community leading to widely accepted and well-formed software engineering processes in the field.

https://github.com/emsejournal/openscience

Reproducible Science is good. Replicated Science is better.

ReScience C is a platinum open-access peer-reviewed journal that targets computational research and encourages the explicit replication of already published research, promoting new and open-source implementations in order to ensure that the original research is reproducible. You can read about the ideas behind ReScience C in the article Sustainable computational science: the ReScience initiative

https://rescience.github.io/

https://reproducible-research.inria.fr/

Software Heritage

GitLab
Reproducible science

“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”

Socio-technical issues: open science, open source software, multi-disciplinary collaboration, incentives/rewards, initiatives, etc.

with many challenges related to data acquisition, knowledge organization/sharing, etc.
“The build process of a software product is reproducible if, after designating a specific version of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed.”
“Computation-driven research really has changed in the last 10 years, driven by three principles of data science, which, after longstanding partial efforts, are finally available in mature form for daily practice, as frictionless open services offering data sharing, code sharing, and competitive challenges.”

[FR-1: Data] + [FR-2: Re-execution] + [FR-3: Challenges]

“We are entering an era of frictionless research exchange, in which research algorithmically builds on the digital artifacts created by earlier research, and any good ideas that are found get spread rapidly, everywhere. The collective behavior induced by frictionless research exchange is the emergent superpower driving many events that are so striking today.”
Frictionless reproducibility

[FR-1: Data] “Datafication of everything, with a culture of research data sharing.”

[FR-2: Re-execution (code)]: “Research code sharing including the ability to exactly re-execute the same complete workflow by different researchers.”

[FR-3: Challenges] “a shared public dataset, a prescribed and quantified task performance metric, a set of enrolled competitors seeking to outperform each other on the task, and a public leaderboard.”
Frictionless reproducibility

[FR-1: Data] “Datafication of everything, with a culture of research data sharing.”

[FR-2: Re-execution (code)]: “Research code sharing including the ability to exactly re-execute the same complete workflow by different researchers.”

[FR-3: Challenges] “a shared public dataset, a prescribed and quantified task performance metric, a set of enrolled competitors seeking to outperform each other on the task, and a public leaderboard.”

frictionless reproducibility = [FR-1] + [FR-2] + [FR-3]

<table>
<thead>
<tr>
<th>If we only have...</th>
<th>We are blocked, because</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[FR-1] + [FR-2]</td>
<td>No defined task</td>
<td>Exploratory Data Analysis</td>
</tr>
<tr>
<td>[FR-1] + [FR-3]</td>
<td>Can’t build on code of others</td>
<td>Netflix Challenge; DARPA Biometric Challenges</td>
</tr>
<tr>
<td>[FR-2] + [FR-3]</td>
<td>No Common Dataset</td>
<td>Human Subjects Clinical Research</td>
</tr>
</tbody>
</table>

Table 1: Leave-One-outs, and what is blocked
Frictionless reproducibility = [FR-1: Data] + [FR-2: Re-execution] + [FR-3: Challenges]

[FR-1] and [FR-2] are quite “standard” but do not come without frictions – more soon! [FR-3] is an important and original piece.

On the one hand, [FR-3] is a way to objectively assess a contribution, compare solutions, and measure progress (if any). [FR-3] sounds legit to provide a “task definition that formalized a specific research problem and made it an object of study”. [FR-3] is “the competitive element that attracted our attention in the first place”.

Think about the absence of [FR-3]. The “challenge paradigm” is a big ongoing shift (see Isabelle Guyon and Evelyne Viegas - "AI Competitions and the Science Behind Contests")

- Many success stories (mainly in empirical machine learning): speech processing, biometric recognition, facial recognition, protein structure prediction problem (CASP), etc.
- More and more leaderboard (eg https://evalplus.github.io/leaderboard.html https://robustbench.github.io/) or competition (eg SAT competition)
- Many platforms, services, and events supporting the shift (eg Kaggle)
Frictionless reproducibility = [FR-1: Data] + [FR-2: Re-execution] + [FR-3: Challenges]

[FR-1] and [FR-2] are quite “standard” but do not come without frictions – more soon! [FR-3] is an important and original piece

On the one hand, [FR-3] is a way to objectively assess a contribution, compare solutions, and measure progress (if any). [FR-3] sounds legit to provide a “task definition that formalized a specific research problem and made it an object of study”. [FR-3] is “the competitive element that attracted our attention in the first place”. The performance measurement crystallized a specific project’s contribution, boiling down an entire research contribution essentially to a single number, which can be reproduced. Think about the absence of [FR-3]

The “challenge paradigm” is a big ongoing shift (see Isabelle Guyon and Evelyne Viegas - "AI Competitions and the Science Behind Contests")

- Many success stories (mainly in empirical machine learning): speech processing, biometric recognition, facial recognition, protein structure prediction problem (CASP), etc.
- More and more leaderboard (eg https://evalplus.github.io/leaderboard.html https://robustbench.github.io/) or competition (eg SAT competition)
- Many platforms, services, and events supporting the shift (eg Kaggle)
Frictionless reproducibility = [FR-1: Data] + [FR-2: Re-execution] + [FR-3: Challenges]

[FR-1] and [FR-2] are quite “standard” but do not come without frictions – more soon! [FR-3] is an important but discussable piece

On the other hand, we know that the power of a simple scoring function is dangerous (e.g., Goodhart's law)

“What if the metric is wrong? What if the subtleties of a complex problem are not amenable to representation by a single scalar? What happens when metrics for locally optimal solutions are apparent, but ones for globally optimal solutions are not? What happens when the community is not (yet) mature enough to rally around a consensus-scoring function? I think it is important to recognize that finding an appropriate scoring function, let alone an objectively best one, is an ongoing task and might evolve as FR-1 and FR-2 provide a deeper understanding of the problem space.”

Overcoming Potential Obstacles as We Strive for Frictionless Reproducibility by Adam D. Schuyler (2024)
Are we frictionless?

Reading a paper in 2024 is sometimes like in 1970:

- Where is the source code? (eg implementation of the solution, scripts to compute metrics)
- Where is the data? (eg to test the solution)
- Contacting authors?
  - no response?
  - code not consistent with the PDF
  - ...
- It does not work on my machine; results are completely different...

There are lots of socio-technical frictions... even when you have the code and data!

=> When people can replicate an analysis at the touch of a button using open-source software, open data, and methods to assess and compare proposals, it significantly eases verification of results, engagement with a diverse range of contributors, and progress
Frictionless reproducibility (an example)

Cutting through buggy adversarial example defenses:
fixing 1 line of code breaks SABRE

Nicholas Carlini
Google DeepMind

Abstract

SABRE is a defense to adversarial examples that was accepted at IEEE S&P 2024. We first reveal significant flaws in the evaluation that point to clear signs of gradient masking. We then show the cause of this gradient masking: a bug in the original evaluation code. By fixing a single line of code in the original repository, we reduce SABRE’s robust accuracy to 0%. In response to this, the authors modify the defense and introduce a new defense component not described in the original paper. But this fix contains a second bug; modifying one more line of code reduces robust accuracy to below baseline levels. After we released the first version of our paper online, the authors introduced another change to the defense; by commenting out one line of code during attack we reduce the robust accuracy to 0% again.

Submission history
From: Nicholas Carlini [view email]
[v1] Mon, 6 May 2024 17:48:24 UTC (19 KB)
[v2] Mon, 27 May 2024 17:41:06 UTC (20 KB)
“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”

Despite the availability of data and code, several studies report that the same data analyzed with different software can lead to different results.
Replicability of the EC-Earth3 Earth system model under a change in computing environment

François Massonnet\textsuperscript{1,2}, Martin Ménégoz\textsuperscript{2,3}, Mario Acosta\textsuperscript{2}, Xavier Yepes-Arbós\textsuperscript{2}, Eleftheria Exarchou\textsuperscript{2}, and Francisco J. Doblas-Reyes\textsuperscript{2,4}

Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one “non-replicable” case (see below) and one replicable case.
Replicability of the EC-Earth3 Earth system model under a change in computing environment

François Massonnet1,2, Martin Ménégoz3,4, Mario Acosta2, Xavier Yepes-Arbós2, Eleftheria Exarchou2, and Francisco J. Doblas-Reyes2,4

Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one “non-replicable” case (see below) and one replicable case.
Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one “non-replicable” case (see below) and one replicable case.

### Table 1. The two computing environments considered in this study.

<table>
<thead>
<tr>
<th>Computing environment</th>
<th>ECMWF-CCA</th>
<th>MareNostrum3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Reading, UK</td>
<td>Barcelona, Spain</td>
</tr>
<tr>
<td>Motherboard</td>
<td>Cray XC30 system</td>
<td>IBM dx360 M4</td>
</tr>
<tr>
<td>Processor</td>
<td>Dual 12-core E5-2697 v2 (Ivy Bridge) series processors (2.7 GHz), 24 cores per node</td>
<td>2x Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 GHz, 16 cores per node</td>
</tr>
<tr>
<td>Operating system</td>
<td>Cray Linux Environment (CLE) 5.2</td>
<td>Linux – SuSe distribution 11 SP2</td>
</tr>
<tr>
<td>Compiler</td>
<td>Intel(R) 64 Compiler XE for applications running on Intel(R) 64, version 14.0.1.106 build 20131008</td>
<td>Intel(R) 64 Compiler XE for applications running on Intel(R) 64, version 13.0.1.117 build 20121010</td>
</tr>
<tr>
<td>MPI version</td>
<td>Cray mpich2 v6.2.0</td>
<td>Intel MPI v4.1.3.049</td>
</tr>
<tr>
<td>LAPACK version</td>
<td>Cray libsci v12.2.0</td>
<td>Intel MKL v11.0.1</td>
</tr>
<tr>
<td>SZIP, HDF5, NetCDF4</td>
<td>v2.1, v1.8.11, v4.3.0</td>
<td>v2.1, v1.8.14, v4.2</td>
</tr>
<tr>
<td>GribAPI, GribEX</td>
<td>v1.13.0, v000395</td>
<td>v1.14.0, v000370</td>
</tr>
</tbody>
</table>

### Table 2. The four experiments considered in this study.

<table>
<thead>
<tr>
<th>Experiment ID</th>
<th>e011</th>
<th>m06e</th>
<th>a0gi</th>
<th>a0go</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing environment</td>
<td>ECMWF-CCA</td>
<td>MareNostrum3</td>
<td>ECMWF-CCA</td>
<td>MareNostrum3</td>
</tr>
<tr>
<td>EC-Earth version</td>
<td>3.1</td>
<td>3.1</td>
<td>3.2</td>
<td>3.2</td>
</tr>
<tr>
<td>Processors (IFS+NEMO+OASIS)</td>
<td>598</td>
<td>512</td>
<td>432 (288 + 144)</td>
<td>416 (288 + 128)</td>
</tr>
<tr>
<td>F flags</td>
<td>(480 + 96 + 22)</td>
<td>(384 + 96 + 22)</td>
<td>(OASIS: library)</td>
<td>(OASIS:library)</td>
</tr>
<tr>
<td>-02g</td>
<td>traceback</td>
<td>-02g</td>
<td>traceback</td>
<td>-02g</td>
</tr>
<tr>
<td>-vec-report0* -r8</td>
<td>-vec-report0* -r8</td>
<td>-vec-report0* -r8</td>
<td>-vec-report0* -r8</td>
<td></td>
</tr>
<tr>
<td>-vec-report0* -r8</td>
<td>xHost</td>
<td>FP-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td></td>
</tr>
<tr>
<td>C flags</td>
<td>-02g</td>
<td>-02g</td>
<td>-02g</td>
<td>-02g</td>
</tr>
<tr>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td></td>
</tr>
<tr>
<td>-fp-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td></td>
</tr>
<tr>
<td>LD flags</td>
<td>-02g</td>
<td>-02g</td>
<td>-02g</td>
<td>-02g</td>
</tr>
<tr>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td></td>
</tr>
<tr>
<td>-fp-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td>-fp-model strict xHost</td>
<td></td>
</tr>
<tr>
<td>Output size</td>
<td>141.8 GB</td>
<td>141.6 GB</td>
<td>101.3 GB</td>
<td>101.3 GB</td>
</tr>
</tbody>
</table>
Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results?

A study involving eight institutions and seven different supercomputers in Europe is currently ongoing with EC-Earth. This ongoing study aims to do the following:

- evaluate different **computational environments** that are used in collaboration to produce CMIP6 experiments (can we safely create large ensembles composed of subsets that emanate from different partners of the consortium?);
- detect if the same **CMIP6 configuration** is replicable among platforms of the EC-Earth consortium (that is, can we safely exchange restarts with EC-Earth partners in order to initialize simulations and to avoid long spin-ups?); and
- systematically evaluate the impact of **different compilation flag options** (that is, what is the highest acceptable level of optimization that will not break the replicability of EC-Earth for a given environment?).
Should software version numbers determine science?

Significant differences were revealed between FreeSurfer version v5.0.0 and the two earlier versions. [...] About a factor two smaller differences were detected between Macintosh and Hewlett-Packard workstations and between OSX 10.5 and OSX 10.6. The observed differences are similar in magnitude as effect sizes reported in accuracy evaluations and neurodegenerative studies.

“Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and executed.”

Reproducibility of neuroimaging analyses across operating systems, Glatard et al., Front. Neuroinform., 24 April 2015

The implementation of mathematical functions manipulating single-precision floating-point numbers in libmath has evolved during the last years, leading to numerical differences in computational results. While these differences have little or no impact on simple analysis pipelines such as brain extraction and cortical tissue classification, their accumulation creates important differences in longer pipelines such as the subcortical tissue classification, RSfMRI analysis, and cortical thickness extraction.
Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and executed.

Reproducibility of neuroimaging analyses across operating systems, Glatard et al., Front. Neuroinform., 24 April 2015

<table>
<thead>
<tr>
<th>Cluster A</th>
<th>Cluster B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Applications</td>
</tr>
<tr>
<td>Freesurfer 5.3.0, build 1</td>
<td>Freesurfer 5.3.0, build 1 and 2</td>
</tr>
<tr>
<td>FSL 5.0.6, build 1</td>
<td>FSL 5.0.6, build 1 and 2</td>
</tr>
<tr>
<td>CIVET 1.1.12-UCSF, build 1</td>
<td>CIVET 1.1.12-UCSF, build 1</td>
</tr>
<tr>
<td>Interpreters</td>
<td>Interpreters</td>
</tr>
<tr>
<td>Python 2.4.3, bash 3.2.25</td>
<td>Python 2.7.5, bash 4.2.47,</td>
</tr>
<tr>
<td>Perl 5.8.8, tcsh 6.14.00</td>
<td>Perl 5.18.2, tcsh 6.18.01</td>
</tr>
<tr>
<td>glibc version 2.5</td>
<td>2.18</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 5.10</td>
</tr>
<tr>
<td>Hardware</td>
<td>Fedora 20</td>
</tr>
<tr>
<td>x86_64 CPUs (Intel Xeon)</td>
<td>x86_64 CPUs (Intel Xeon)</td>
</tr>
</tbody>
</table>

Statically building programs improves reproducibility across OSes, but small differences may still remain when dynamic libraries are loaded by static executables [...]. When static builds are not an option, software heterogeneity might be addressed using virtual machines. However, such solutions are only workarounds: differences may still arise between static executables built on different OSes, or between dynamic executables executed in different VMs.
Reproducible science as a (deep) software variability problem

“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”

Despite the availability of data and code, several studies report that the same data analyzed with different software can lead to different results.
Despite the availability of data and code, several studies report that the same data analyzed with different software can lead to different results. Many layers (operating system, third-party libraries, versions, workloads, compile-time options and flags, etc.) themselves subject to variability can alter the results.

Reproducible science and deep software variability: a threat and opportunity for scientific knowledge!
How often \((x+y)+z = x+(y+z)\) ?

https://github.com/FAMILIAR-project/reproducibility-associativity/
Frictionless Reproducibility and (Deep) Software (Variability)

Problem (cont’d): Variability and Frictions

Solution: Variability and Exploration

Discussions
15,000+ options
thousands of compiler flags and compile-time options
dozens of preferences
100+ command-line parameters
1000+ feature toggles

hardware variability

Non-functional properties
execution time
energy consumption
security
accuracy
15,000+ options

thousands of compiler flags and compile-time options

dozens of preferences

input data

100+ command-line parameters

1000+ feature toggles

deep software variability

System under Study (reproducible and replicable)

Variability Output (scientific result; most of the time quantitative information)
Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one “non-replicable” case (see below) and one replicable case.

Table 1. The two computing environments considered in this study.

<table>
<thead>
<tr>
<th>Computing environment</th>
<th>ECMWF-CCA</th>
<th>MareNostrum3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Reading, UK</td>
<td>Barcelona, Spain</td>
</tr>
<tr>
<td>Motherboard</td>
<td>Clay X30 system</td>
<td>IBM i396 M4</td>
</tr>
<tr>
<td>Processor</td>
<td>Dual 3-core ES-2697 v2 (Ivy Bridge) series processors (2.7 GHz), 24 cores per node</td>
<td>2x Intel Xeon E5-2670 v4 (Broadwell EP) 2.3 GHz, 16 cores per node</td>
</tr>
<tr>
<td>Operating system</td>
<td>Clay Linux Environment (CLE) 5.2</td>
<td>Linux - Suse distribution 11 SP2</td>
</tr>
<tr>
<td>Compiler</td>
<td>Intel(R) 64 Compiler XE for applications running on Intel(R)64, version 14.0.1.106 build 20131006</td>
<td>Intel(R) 64 Compiler XE for applications running on Intel(R)64, version 14.0.1.117 build 20121010</td>
</tr>
<tr>
<td>MPI version</td>
<td>Clay mpix2 v2.2.0</td>
<td>Intel MPI v3.1.0.49</td>
</tr>
<tr>
<td>LAPACK version</td>
<td>Clay ibsci v2.2.0</td>
<td>Intel MKL v11.0.1</td>
</tr>
<tr>
<td>ZIP, HDF5, NetCDF4</td>
<td>v2.1, v8.11, v8.3.0</td>
<td>v2.1, v8.14, v4.2</td>
</tr>
<tr>
<td>GrbAPI, GrbEX</td>
<td>v1.13.0, v000095</td>
<td>v1.14.0, v000370</td>
</tr>
</tbody>
</table>

Table 2. The four experiments considered in this study.

<table>
<thead>
<tr>
<th>Experiment ID</th>
<th>e01</th>
<th>m06e</th>
<th>e01p</th>
<th>a01p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing environment</td>
<td>ECMWF-CCA</td>
<td>MareNostrum3</td>
<td>ECMWF-CCA</td>
<td>MareNostrum3</td>
</tr>
<tr>
<td>EC-Earth version</td>
<td>3.1</td>
<td>3.1</td>
<td>3.2</td>
<td>3.2</td>
</tr>
<tr>
<td>Processes (IFS+NEMO+OASIS)</td>
<td>(401 + 96 + 22)</td>
<td>(344 + 96 + 22)</td>
<td>(432 (288 + 144) (OASIS library)</td>
<td>(462 (288 + 128) (OASIS library)</td>
</tr>
<tr>
<td>F flags</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
</tr>
<tr>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
</tr>
<tr>
<td>-vec-report0 =8</td>
<td>-vec-report0 =8</td>
<td>-vec-report0 =8</td>
<td>-vec-report0 =8</td>
<td>-vec-report0 =8</td>
</tr>
<tr>
<td>C flags</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
</tr>
<tr>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
</tr>
<tr>
<td>LD flags</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
</tr>
<tr>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
<td>-traceback</td>
</tr>
<tr>
<td>Output size</td>
<td>141.8 GB</td>
<td>141.6 GB</td>
<td>101.3 GB</td>
<td>101.3 GB</td>
</tr>
</tbody>
</table>
We demonstrate that effects of parameter, hardware, and software variation are detectable, complex, and interacting. However, we find most of the effects of parameter variation are caused by a small subset of parameters. Notably, the entrainment coefficient in clouds is associated with 30% of the variation seen in climate sensitivity, although both low and high values can give high climate sensitivity. We demonstrate that the effect of hardware and software is small relative to the effect of parameter variation and, over the wide range of systems tested, may be treated as equivalent to that caused by changes in initial conditions.

57,067 climate model runs. These runs sample parameter space for 10 parameters with between two and four levels of each, covering 12,487 parameter combinations (24% of possible combinations) and a range of initial conditions.
Deep Reinforcement Learning that Matters

Peter Henderson¹*, Riashat Islam¹,²*, Philip Bachman²
Joelle Pineau¹, Doina Precup¹, David Meger¹

What is the magnitude of the effect hyperparameter settings can have on baseline performance?

How does the choice of network architecture for the policy and value function approximation affect performance?

How can the reward scale affect results?

Can random seeds drastically alter performance?

How do the environment properties affect variability in reported RL algorithm performance?

Are commonly used baseline implementations comparable?
“Completing a full replication study of our previously published findings on bluff-body aerodynamics was harder than we thought. Despite the fact that we have good reproducible-research practices, sharing our code and data openly.”

Story 1: Meshing and boundary conditions can ruin everything

Story 3: All linear algebra libraries are not created equal

Story 4: Different versions of your code, external libraries or even compilers may challenge reproducibility
Data analysis workflows in many scientific domains have become increasingly complex and flexible (subject to variability). Here we assess the effect of this flexibility on the results of functional magnetic resonance imaging by asking 70 independent teams to analyse the same dataset, testing the same 9 ex-ante hypotheses. The flexibility of analytical approaches is exemplified by the fact that no two teams chose identical workflows to analyse the data. This flexibility resulted in sizeable variation in the results of hypothesis tests, even for teams whose statistical maps were highly correlated at intermediate stages of the analysis pipeline. Variation in reported results was related to several aspects of analysis methodology. Notably, a meta-analytical approach that aggregated information across teams yielded a significant consensus in activated regions. Furthermore, prediction markets of researchers in the field revealed an overestimation of the likelihood of significant findings, even by researchers with direct knowledge of the dataset. Our findings show that analytical flexibility can have substantial effects on scientific conclusions, and identify factors that may be related to variability in the analysis of functional magnetic resonance imaging. The results emphasize the importance of validating and sharing complex analysis workflows, and demonstrate the need for performing and reporting multiple analyses of the same data. Potential approaches that could be used to mitigate issues related to analytical variability are discussed.
Can Machine Learning Pipelines Be Better Configured? Wang et al. FSE’2023

“A pipeline is subject to misconfiguration if it exhibits significantly inconsistent performance upon changes in the versions of its configured libraries or the combination of these libraries. We refer to such performance inconsistency as a pipeline configuration (PLC) issue.”

In this paper, we empirically studied 11,363 ML pipelines from diverse competitions on KAGGLE to explore the impacts of different ML library version combinations on their performances. Our study reveals the pervasiveness and severity of PLC issues in ML pipelines. Our findings can motivate the establishment of a symbiotic ecosystem where researchers, tool builders, and library vendors work together to assist developers in combating PLC issues.
Deep software variability: Are layers/features orthogonal or are there interactions?

Configuration is hard: numerous options, informal knowledge

-bframes 1 --ref 3 --cabac DiverSE-meeting.mp4 -o meeting13.webm

mathieuacher localhost.localdomain ~ x264 --fullhelp | wc -l

Lossless:
- x264 --qp 0 -o <output> <input>

Maximum PSNR at the cost of speed and visual quality:
- x264 --preset placebo --tune psnr -o <output> <input>

Constant bitrate at 1000kbps with a 2 second-buffer:
- x264 --vbv-bfs 2000 --vbv-bfu 1000 -o <output> <input>

Presets:
- -profile <string> Force the limits of an H.264 profile
- -preset <string> Use a preset to select encoding settings
- -tune <string> Tune the settings for a particular type of source

Ratecontrol:
- -b, --bitrate <integer> Set bitrate (kbit/s)
- -crf <float> Quality-based VBR
- -vbv-maxrate <integer> Max local bitrate
- -vbv-bfs <integer> Set size of the VBR buffer
- -p, --pass <integer> Enable multipass encoding

-I, --keyint {integer or "infinite"} Maximum GOP size [250]
- -ttf Encryption (true or false)
- --bfr Use b frame prediction
- --pulldown <string> Use soft pulldown (false or true)
REAL WORLD EXAMPLE (x264)

**Hardware**: Dell Latitude 7400, Raspberry Pi 4 model B

**Operating System**: Ubuntu 20.04, Raspberry Pi 10.4

**Software**:
- x264 --mbtree
- x264 --no-mbtree

**Input Data**:
- Duration (s): 6, 22, 6, 25, 73, 351, 72, 359
- Size (MB): 33, 28, 21, 34, 33, 28, 21, 34

**Animation Types**:
- vertical, animation, vertical, animation, vertical, animation, vertical, animation

** REAL WORLD Example (x264)**
# REAL WORLD EXAMPLE (x264)

## Hardware
- Dell Latitude 7400
- Raspberry Pi 4 Model B

## Operating System
- Ubuntu

## Software
- x264 --mbtree
- x264 --no-mbtree

## Input Data
- Animation
- Vertical

<table>
<thead>
<tr>
<th></th>
<th>Duration (s)</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>351</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>359</td>
<td>34</td>
</tr>
</tbody>
</table>
REAL WORLD EXAMPLE (x264)

**Hardware**
- Dell Latitude 7400
- Raspberry Pi 4 Model B

**Operating System**
- Ubuntu 20.04
- Ubuntu 10.4

**Software**
- x264 --mbtree
- x264 --no-mbtree

**Input Data**
- Animation
- Vertical

<table>
<thead>
<tr>
<th>Duration (s)</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>25</td>
<td>34</td>
</tr>
<tr>
<td>73</td>
<td>33</td>
</tr>
<tr>
<td>351</td>
<td>28</td>
</tr>
<tr>
<td>72</td>
<td>21</td>
</tr>
<tr>
<td>359</td>
<td>34</td>
</tr>
</tbody>
</table>

≈ *16
≈ *12
The “best”/default software variant might be a bad one.

Influential software options and their interactions vary.

Performance prediction models and variability knowledge may not generalize.

---

Let’s go deep with input data!

Intuition: video encoder behavior (and thus runtime configurations) hugely depends on the input video (different compression ratio, encoding size/type etc.)

Is the best software configuration still the best?
Are influential options always influential?
Does the configuration knowledge generalize?

\[ f_1 = \beta \times f_2 + \alpha \]

YouTube User General Content dataset: 1397 videos
Measurements of 201 soft. configurations (with same hardware, compiler, version, etc.): encoding time, bitrate, etc.
**Inputs =**

### configurations' measurements over input_1

<table>
<thead>
<tr>
<th>configurationID</th>
<th>cabac</th>
<th>ref</th>
<th>mixed_ref</th>
<th>me</th>
<th>subme</th>
<th>me_range</th>
<th>trellis</th>
<th>elapsedtime</th>
<th>fps</th>
<th>rank_elapsedtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:14</td>
<td>375.22</td>
<td>1</td>
</tr>
<tr>
<td>138</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>tesa</td>
<td>10</td>
<td>24</td>
<td>2</td>
<td>0:04:54</td>
<td>155.35</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:22</td>
<td>384.22</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:24</td>
<td>375.4</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>hex</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:19</td>
<td>385.82</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:84</td>
<td>260.65</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:61</td>
<td>303.2</td>
<td>5</td>
</tr>
</tbody>
</table>

### configurations' measurements over input_42

<table>
<thead>
<tr>
<th>configurationID</th>
<th>cabac</th>
<th>ref</th>
<th>mixed_ref</th>
<th>me</th>
<th>subme</th>
<th>me_range</th>
<th>trellis</th>
<th>elapsedtime</th>
<th>fps</th>
<th>rank_elapsedtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>04:37</td>
<td>375.22</td>
<td>3</td>
</tr>
<tr>
<td>138</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>tesa</td>
<td>10</td>
<td>24</td>
<td>2</td>
<td>07:56</td>
<td>155.35</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>07:23</td>
<td>384.22</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>04:33</td>
<td>375.4</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>hex</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>06:00</td>
<td>385.92</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>05:48</td>
<td>260.65</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>02:19</td>
<td>303.2</td>
<td>1</td>
</tr>
</tbody>
</table>
### Inputs =

#### Configurations' measurements over input_1

<table>
<thead>
<tr>
<th>configurationID</th>
<th>cabac</th>
<th>ref</th>
<th>mixed_ref</th>
<th>me</th>
<th>subme</th>
<th>me_range</th>
<th>trellis</th>
<th>elapsed_time</th>
<th>tps</th>
<th>rank_elapsed_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:14</td>
<td>375.22</td>
<td>1</td>
</tr>
<tr>
<td>136</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>tessa</td>
<td>10</td>
<td>24</td>
<td>2</td>
<td>0:04:54</td>
<td>155.35</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:22</td>
<td>384.22</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:24</td>
<td>375.4</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>hex</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:19</td>
<td>385.92</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:04</td>
<td>260.65</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0:02:01</td>
<td>303.2</td>
<td>5</td>
</tr>
</tbody>
</table>

#### Generalization/transfer:

What's the relationship between perf_pred_1 and perf_pred_42?

- With perf_pred_i a performance model capable of predicting performance of any configuration on input_i linear relationship?
  - Eg Pearson/Spearman linear correlation
- Influential features/options: same?

#### Configurations' measurements over input_42

<table>
<thead>
<tr>
<th>configurationID</th>
<th>cabac</th>
<th>ref</th>
<th>mixed_ref</th>
<th>me</th>
<th>subme</th>
<th>me_range</th>
<th>trellis</th>
<th>elapsed_time</th>
<th>tps</th>
<th>rank_elapsed_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0:03:37</td>
<td>375.22</td>
<td>3</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>tessa</td>
<td>10</td>
<td>24</td>
<td>2</td>
<td>0:07:56</td>
<td>320.35</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0:07:23</td>
<td>284.72</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0:04:33</td>
<td>375.4</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>hex</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0:06:00</td>
<td>285.92</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0:05:48</td>
<td>260.65</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>dia</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0:02:19</td>
<td>303.2</td>
<td>1</td>
</tr>
</tbody>
</table>
Let’s go deep with **input data!**

Intuition: video encoder behavior (and thus **runtime configurations**) hugely depends on the **input video** (different compression ratio, encoding size/type etc.)

Is the best software configuration still the best?
Are influential options always influential?
Does the configuration knowledge generalize?

\[ f_1 = \beta \times f_2 + \alpha \]

YouTube User General Content dataset: **1397 videos**
Measurements of **201 soft. configurations** (with same hardware, compiler, version, etc.): encoding time, bitrate, etc.
Do x264 software performances stay consistent across inputs?

- **Encoding time**: very strong correlations
  - low input sensitivity

- **FPS**: very strong correlations
  - low input sensitivity

- **CPU usage**: moderate correlation, a few negative correlations
  - medium input sensitivity

- **Bitrate**: medium-low correlation, many negative correlations
  - High input sensitivity

- **Encoding size**: medium-low correlation, many negative correlations
  - High input sensitivity

Two performance models $f_1$ and $f_2$: $f_1 = \beta \times f_2 + \alpha$
Are there some configuration options more sensitive to input videos? (bitrate)
Are there some configuration options more sensitive to input videos? (bitrate)
Practical impacts for users, developers, scientists, and self-adaptive systems

Threats to variability knowledge: predicting, tuning, or understanding configurable systems without being aware of inputs can be inaccurate and... pointless

Opportunities: for some performance properties (P) and subject systems, some stability is observed and performance remains consistent!

Sometimes, variability is consistent/stable and knowledge transfer is immediate. But there are also interactions among variability layers and variability knowledge may not generalize.
Does deep software variability affect previous scientific, software-based studies? (A graphical template)

List all details... and questions:

What if we run the experiments on different:

OS?

Version/Commit?

Parameters?

Input?
AGENDA

Frictionless Reproducibility and (Deep) Software (Variability)

Problem: Variability and Frictions

Solution: Variability and Exploration

Discussions
Deep variability problem (statement)

Fundamentally, we have a huge multi-dimensional variant space (eg $10^{6000}$)

run (source_code) => result

run (hardware, operating_system, build_environment, input_data, source_code, …) => results

Fixing variability once and for all, in all dimensions/layers, is the obvious solution...

But it is either impossible (eg the ages of processor can have an impact on execution time)...

Or not desirable

● non-robust result
● generalization/transferability of the results/findings
● kill innovation
Replicability is the holy grail!

Exploring various configurations:

- Make more robust scientific findings
- Define and assess the validity enveloppe
- Enable exploration and optimization
- Innovation and new hypothesis, insights, knowledge

⇒ We propose to embrace deep variability for the sake of replicability
Embrace deep variability!

Explicit modeling of the variability points and their relationships, such as:

1. Get insights into the variability “factors” and their possible interactions
2. Capture and document configurations for the sake of **reproducibility**
3. Explore diverse configurations to **replicate**, and hence optimize, validate, increase the robustness, or provide better resilience

⇒ We aim to **address the complexities associated with reproducibility and replicability in modern software systems and environments**, facilitating a more comprehensive and nuanced perspective on these critical “factors”.

**Our Vision**

**Embracing Deep Variability For Reproducibility & Replicability**

Mathieu Acher, Benoît Combemel, Georges Aaron Randriamaina, Jean-Marc Jézéquel

IRISA, Université de Rennes
Rennes, France

**ABSTRACT**

Reproducibility (a.k.a., determinism in some cases) constitutes a fundamental aspect in various fields of computer science, such as floating-point computations in numerical analysis and simulation, concurrency models in parallelism, reproducible builds for third-party integration and packaging, and containerization for execution environments. These concepts, while pervasive across diverse concerns, often exhibit intricate inter-dependencies, making it challenging to achieve a comprehensive understanding. In this short paper, we delve into the application of software engineering techniques, specifically variability management, to systematically identify and explicits points of variability that may give rise to reproducibility issues (e.g., language, libraries, compiler, virtual machine, OS, environment variables, etc.). The primary objectives are: i) gaining insights into the variability layers and their possible interactions, ii) capturing and documenting configurations for the sake of reproducibility, and iii) exploring diverse configurations to replicate, and hence validate and ensure the robustness of results. By adopting these methodologies, we aim to address the complexities associated with reproducibility and replicability in modern software systems and environments, facilitating a more comprehensive and nuanced perspective on these critical aspects.

In this paper, we propose to characterize both intended and unintended variability of any software-intensive system in order to support reproducibility and replicability, and eventually estimate its robustness, uncertainty profile, and explore different pathways.

**2 DEEP SOFTWARE VARIABILITY**

Uncertainty in informatics comes from many different origins [16, 36], either ontological (i.e., inherent unpredictability, e.g., stochastic) or epistemic (i.e., due to insufficient knowledge).

Ontological causes include noise in the input data of a program, its memory layout, network delays, the internal state of the processor, the ambient temperature, and even the age of the processor 4.

Epistemic causes include misunderstanding of the user’s needs, variable behavior of conceptually similar resolution methods, choice of threshold parameters, unexpected behavior of APIs, variable behavior among functionally similar libraries, or subtle differences in the semantics of programming languages (e.g., -392 evaluates to -1 in Java but to 1 in Python), or even inside the same programming language (for instance if x is an undefined behavior in C).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Input Data</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g., random seed selection</td>
<td>e.g. x=42 vs. x=22 vs. x=12</td>
<td>e.g., random seed selection</td>
</tr>
</tbody>
</table>
Solution #1: Variability model

- Abstractions are definitely needed to...
  - reason about logical constraints and interactions
  - integrate domain knowledge
  - synthesize domain knowledge
  - automate and guide the exploration of variants
  - scope and prioritize experiments

- Language and formalism: **feature model** (widely applicable!)
  - translation to logics
  - reasoning with SAT/CP/SMT solvers
Solution #1: Variability model

- Abstractions are definitely needed…
- Yes, but how to obtain a feature model?
  - modelling
  - reverse engineering (out of command-line parameters, source code, logs, configurations, etc.)
  - learning (next slide!)
  - modeling+reverse engineering+learning (HDR)
Solution #2: sampling and learning
(regression, classification)

Whole Population of Configurations → Training Sample → Performance Measurements → Prediction Model

\[ f : \mathbb{C} \rightarrow \mathbb{R}^+ \] the function affecting to any configuration \( c \in \mathbb{C} \) its performance \( f(c) \in \mathbb{R}^+ \),

“Learning Software Configuration Spaces: A Systematic Literature Review” JSS, 2021
x264 --me dia
--ref 5
...
-o output_1.x264
The build process of a software product is reproducible if, after designating a specific version of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed.

Lamb and Zacchirolı “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022
“The build process of a software product is reproducible if, after designating a specific version of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed.” Lamb and Zacchirolı “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022

```
make defconfig # configuration
make # build the kernel (binary) out of config
make # should be the same, right?
```

Table 2: Identified options and their category.

<table>
<thead>
<tr>
<th>Option</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE_SIG_SHA1</td>
<td>Module Signing</td>
</tr>
<tr>
<td>MODULE_SIG_SHA224</td>
<td>Module Signing</td>
</tr>
<tr>
<td>MODULE_SIG_SHA256</td>
<td>Module Signing</td>
</tr>
<tr>
<td>MODULE_SIG_SHA384</td>
<td>Module Signing</td>
</tr>
<tr>
<td>MODULE_SIG_SHA512</td>
<td>Module Signing</td>
</tr>
<tr>
<td>MODULE_SIG</td>
<td>Module Signing</td>
</tr>
<tr>
<td>GCOV_PROFILE_FTRACE</td>
<td>Profiling</td>
</tr>
<tr>
<td>GCOV_PROFILE_ALL</td>
<td>Profiling</td>
</tr>
<tr>
<td>DEBUG_INFO_SPLIT</td>
<td>Debug Info</td>
</tr>
<tr>
<td>DEBUG_INFO_REduced</td>
<td>Debug Info</td>
</tr>
</tbody>
</table>

Diagram showing the process of reproducing builds with options and configurations.
#1 take away message: look at every variability layer when you want a bit-to-bit reproducibility; don’t ignore compile-time options!

“The build process of a software product is reproducible if, after designating a specific version and a specific variant of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed.” Lamb and Zacchirolí “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022
Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems Randrianaina, Khelladi, Zendra, Acher MSR’2024

#2 take away message: interactions across variability layers exist (eg compile-time option with build path) and may hamper reproducibility

“The build process of a software product is reproducible if, after designating a specific version and a specific variant of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed.” Lamb and Zacchirolı “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022

Busybox. To pinpoint the source of non-reproducible builds, the typical workflow is to slightly vary the build environment. Changing the build path between two builds of the same configuration for Busybox impacts 49.75% of the configurations, causing their build to be non-reproducible (presented in Figure 2 under the name Busybox (alter)). The decision tree identifies the option involved which is DEBUG. In fact, this option includes some debug information in the binary including the build path. Thus, interactions exist between configuration options and build environment. This can be solved in two ways, either by disabling the option, or not changing the build environment. There, a trade-off is to be made because the developer may need DEBUG and require the build to be reproducible. Building in the same directory solved the issue and the configurations we have picked for Busybox are reproducible at 100% as shown in Figure 2. Overall, altering the build environment in Busybox identified the DEBUG option as key to achieving 100% reproducibility, either by disabling it or maintaining a consistent build path.
Linux as a subject software system (not as an OS interacting with other layers)

Targeted non-functional, quantitative property: **binary size**
- interest for maintainers/users of the Linux kernel (embedded systems, cloud, etc.)
- challenging to predict (cross-cutting options, interplay with compilers/build systems, etc.)

Dataset: version 4.13.3 (September 2017), x86_64 arch, measurements of 95K+ random configurations
- paranoiac about deep variability since 2017, Docker to control the build environment and scale
- diversity of binary sizes: from 7Mb to 1.9Gb
- 6% MAPE errors: quite good, though costly…

### Table I: Dataset properties for each version. The number of deleted/new features, delta commits, files changes are w.r.t. 4.13.

<table>
<thead>
<tr>
<th>Version</th>
<th>Release Date</th>
<th>LOC</th>
<th>Files</th>
<th>Examples</th>
<th>Seconds/config</th>
<th>Options</th>
<th>Features</th>
<th>Deleted features</th>
<th>New features</th>
<th>ΔCommits</th>
<th>Files changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.13</td>
<td>2017/09/03</td>
<td>16,616,534</td>
<td>60,530</td>
<td>92,562</td>
<td>not available</td>
<td>12,776</td>
<td>9,468</td>
<td>-</td>
<td>-</td>
<td>299</td>
<td>31,052</td>
</tr>
<tr>
<td>4.15</td>
<td>2018/01/28</td>
<td>17,073,368</td>
<td>62,249</td>
<td>39,391</td>
<td>not available</td>
<td>12,908</td>
<td>9,425</td>
<td>342</td>
<td>-</td>
<td>299</td>
<td>104,691</td>
</tr>
<tr>
<td>4.20</td>
<td>2018/12/23</td>
<td>17,526,171</td>
<td>62,423</td>
<td>23,489</td>
<td>225</td>
<td>13,533</td>
<td>10,189</td>
<td>468</td>
<td>1,189</td>
<td>118,778</td>
<td>1,972,020</td>
</tr>
<tr>
<td>5.0</td>
<td>2019/03/03</td>
<td>17,679,372</td>
<td>63,076</td>
<td>19,952</td>
<td>247</td>
<td>13,673</td>
<td>10,293</td>
<td>494</td>
<td>1,319</td>
<td>217,935</td>
<td>2,170,935</td>
</tr>
<tr>
<td>5.4</td>
<td>2019/10/24</td>
<td>19,358,903</td>
<td>67,915</td>
<td>25,847</td>
<td>285</td>
<td>14,159</td>
<td>10,813</td>
<td>663</td>
<td>2,008</td>
<td>181,308</td>
<td>3,827,025</td>
</tr>
<tr>
<td>5.7</td>
<td>2020/03/31</td>
<td>19,358,903</td>
<td>67,915</td>
<td>20,159</td>
<td>258</td>
<td>14,586</td>
<td>11,338</td>
<td>715</td>
<td>2,585</td>
<td>225,804</td>
<td>4,393,117</td>
</tr>
<tr>
<td>5.8</td>
<td>2020/08/02</td>
<td>19,729,197</td>
<td>69,303</td>
<td>21,923</td>
<td>289</td>
<td>14,817</td>
<td>11,530</td>
<td>730</td>
<td>2,792</td>
<td>242,381</td>
<td>4,681,313</td>
</tr>
</tbody>
</table>

**Error variation on different versions**

**Hardware**

**Operating System**

**Software**

**Input Data**

---

**Version**

**Input Data**

**Software**

**Operating System**

**Hardware**
Solution #3 Transfer learning (reuse of knowledge)

- Mission Impossible: Saving variability knowledge and prediction model 4.13 (15K hours of computation)
- Heterogeneous transfer learning: the feature space is different
- TEAMS: transfer evolution-aware model shifting

### Solution #3 Transfer learning (con’t)

Luc Lesoil, Helge Spieker, Arnaud Gotlieb, Mathieu Acher, Paul Temple, Arnaud Blouin, Jean-Marc Jézéquel:

#### Figure 1: The performance prediction problem: how to predict software performance considering both configurations and inputs?

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description of the approach</th>
<th>Offline cost (Offshore Organization)</th>
<th>Online measurement cost (User)</th>
<th>Input properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised online learning</td>
<td>Train performance model on demand, from scratch, each time a new input is fed to the configurable system</td>
<td>None</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>Offline learning</td>
<td>Use a pre-trained model over measurements of multiple configurations and inputs. Input properties are used to make the prediction (in an online setting).</td>
<td>High</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Transfer learning</td>
<td>Adapt a pre-trained model for a new targeted input. It requires to gather fresh measurements of some configurations over the input (in an online setting).</td>
<td>Medium</td>
<td>Medium</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Is there an interplay between compile-time and runtime options?

This paper investigates how compile-time options can affect software performances and how compile-time options interact with run-time options.

Figure 1: Cross-layer variability of x264
Solution #4: Leverage stability across variability layers!

First good news: *Worth tuning software at compile-time!*

Second good news: For all the execution time distributions of x264 and all the input videos, the worst correlation is greater than 0.97. If the compile-time options change the scale of the distribution, they do not change the rankings of run-time configurations (i.e., they do not truly interact with the run-time options).

*It has three practical implications:*

1. **Reuse of configuration knowledge:** transfer learning of prediction models boils down to apply a linear transformation among distributions. Users can also trust the documentation of run-time options, consistent whatever the compile-time configuration is.
2. **Tuning at lower cost:** finding the best compile-time configuration among all the possible ones allows one to immediately find the best configuration at run time. We can remove away one dimension!
3. **Measuring at lower cost:** do not use a default compile-time configuration, use the less costly once since it will generalize!

Did we recommend to use two binaries? YES, one for measuring, another for optimal performances!
Key results (for x264)

First good news: Worth to tune software at compile-time!

Second good news: For all the execution time distributions of x264 and all the input videos, the worst correlation is greater than 0.97. If the compile-time options change the scale of the distribution, they do not change the rankings of run-time configurations (i.e., they do not truly interact with the run-time options).

It has three practical implications:

1. **Reuse of configuration knowledge:** transfer learning of prediction models boils down to apply a linear transformation among distributions. Users can also trust the documentation of run-time options, consistent whatever the compile-time configuration is.

2. **Tuning at lower cost:** finding the best compile-time configuration among all the possible ones allows one to immediately find the best configuration at run time. We can remove away one dimension!

3. **Measuring at lower cost:** do not use a default compile-time configuration, use the less costly once since it will generalize!

Did we recommend to use two binaries? YES, one for measuring, another for reaching optimal performances!

What is your move?
What is your prompt?
Petite note : j’ai remarqué que jusqu’à 1800 Elo, sa sensibilité au prompt est un peu bizarre. Par exemple avec “1-0” (Blanc gagne) il joue le coup correct g6. Mais avec “0-1” (Noir gagne) ou “1/2-1/2”, il joue Nf6 (ce qui est illogique vu que Nf6 fait justement perdre les Noirs).

Mais il n’a plus ce comportement bizarre quand le Elo dépasse 2000. Enfin bref ! Il faudrait faire d’autres tests pour déterminer à quel point cette information sur le niveau Elo influence son niveau de jeu, mais c’est déjà intéressant comme petite expérience.
# Optional GM titles
black_title = "[BlackTitle \"GM\"]" if include_black_title else ""
white_title = "[WhiteTitle \"GM\"]" if include_white_title else ""

# Construct the PGN header with the configurable options
pgn_headers = f"\n[Site "Los Angeles, USA"]
[Date "2024.12.01"]
[Round =5]
[White "{white_name}\"]
[Black "{black_name}\""]
[Result "{result}\""]
[WhiteElo "{white_elo}\"]
[BlackElo "{black_elo}\""]
[WhiteTitle]
[BlackTitle]
[TimeControl "40/7200:20/3600:900+30"]
[UTCDate "2024.11.27"]
[UTCTime "09:01:25"]
[Variant "Standard"]
"

# Define possible values for each parameter
results = ["1-0", "1/2-1/2", "0-1"]
# results = ["1-0", "0-1"]
names = ["Nepomniachtchi, Ian", "Kramnik, Vladimir", "Giraud, Thibaut", "Louapre, David", "XXX"]
elos = [1000, 1400, 1700, 1800, 2000, 2900]
# include_title = [True, False]
include_title = [False]

gpt_config = GPTConfig(
    model_gpt="gpt-3.5-turbo-instruct",
    temperature=0.0,
    max_tokens=5,
    chat_gpt=False,
    system_role_message=None  # Since it wasn't provided in the original call
)
Petite note: j’ai remarqué que jusqu’à 1800 Elo, sa sensibilité au prompt est un peu bizarre. Par exemple avec “1-0” (Blanc gagne) il joue le coup correct g6. Mais avec ‘0-1’ (Noir gagne) ou ’½/½/½’, il joue Nf6 (ce qui est illégal vu que Nf6 fait justement perdre les Noirs).

Mais il n’a plus ce comportement bizarre quand le Eo dépasse 2000. Enfin bref ! Il faudrait faire d’autres tests pour déterminer à quel point cette information sur le niveau Eo influence son niveau de jeu, mais c’est déjà intéressant comme petite expérience.
Solution #5: Strategic exploration with modelling and learning

```python
# Construct the PGN header with the configurable options
pgn_headers = f'''
[Site "Los Angeles, USA"]
[Date "2024.12.01"]
[Round "5"]
[White "{white_name}"]
[Black "{black_name}"]
'''

# Define possible values for each parameter
results = ["1-0", "1/2-1/2", "0-1"]
results = ["1-0", "0-1"]
names = ["Nepomniachtchi, Ian", "Kramnik, Vladimir", "Giraud, Thibaut", "Louapre, David", "XXX"]
elos = [1600, 1400, 1700, 1800, 2000, 2900]
# include_title = [True, False]
include_title = [False]
```
Solution #6 Identification of root causes of variability (testing and verification)

Multi-Level Analysis of Compiler-Induced Variability and Performance Tradeoffs

Michael Bentley
Ian Briggs
Ganesh Gopalakrishnan
mbentley@cs.utah.edu
ianbriggs@utah.com
ganesh@cs.utah.edu
University of Utah

Dong H. Ahn
Ignacio Laguna
Gregory L. Lee
Holger E. Jones
ahni@llnl.gov
lagunaperla1@llnl.gov
lee218@llnl.gov
jones19@llnl.gov
Lawrence Livermore National Laboratory

Figure 1: Multi-level workflow. Levels are (1) determine variability-inducing compilations, (2) analyze the space of reproducibility and performance, and (3) debug variability by identifying files and functions causing variability.
I have the following program. I'd like to create variations over this program using conditional compilation. Specifically, I'd like to create a first variation that consists in replacing the type long double by double... and another alternative variation that consists in replacing long double by float.

```c
#include <stdio.h>

int main() {
    // Declare and initialize variables
    long double x = 1.0;
    long double y = 0x1p-53 + 0x1p-64;
    long double z = x + y;

    // Print the value of z
    printf("z = \n", z);
    return 0;
}
```

**Solution #7: LLMs to support exploration of variants space**
Write a Python script that compiles executes the three variants above with and without the `-mfpmath=387` of gcc... Compile also with clang. Synthesize all results in a CSV containing the configuration (flag, with or without `-mfpmath=387`, compiler) and the different outputs. Please use pandas for writing the CSV... also the fieldnames should be compiler, flags, type, and result. The C file to compile is assoc.c

We then refine and ask:

Can you think about gcc flags or clang flags that could be combined/used and have an influence on floating point operations? Please implement them in the Python code.
def compile_and_run(compiler, flags):
    commands = [
        f"{compiler} {flags} -o original approx.c",
        f"{compiler} {flags} -DUSE_DOUBLE -o double_variation approx.c",
        f"{compiler} {flags} -DUSE_FLOAT -o float_variation approx.c",
    ]

    results = []
    for cmd in commands:
        subprocess.run(cmd, shell=True, check=True)

    binary = cmd.split('/')[-2]
    output = subprocess.check_output(f"./{binary}", shell=True).decode().strip()
    results.append(output)

    return results

def main():
    configurations = [
        ('gcc', ''),
        ('gcc', '-mfpmath=387'),
        ('gcc', '-ffast-math'),
        ('gcc', '-fno-signed-zeros'),
        ('gcc', '-fno-trapping-math'),
        ('gcc', '-fno-rounding-math'),
        ('clang', ''),
        ('clang', '-mno-sse'),
        ('clang', '-ffast-math'),
        ('clang', '-fno-signed-zeros'),
        ('clang', '-fno-trapping-math'),
        ('clang', '-fno-rounding-math'),
    ]

    data = []
    for compiler, flags in configurations:
        results = compile_and_run(compiler, flags)
        data.extend([
            {'compiler': compiler, 'flags': flags, 'type': 'original', 'result': results[0]},
            {'compiler': compiler, 'flags': flags, 'type': 'double', 'result': results[1]},
            {'compiler': compiler, 'flags': flags, 'type': 'float', 'result': results[2]},
        ])
Some solutions

- abstractions/models
- learning and sampling
- reuse of configuration knowledge
- leveraging stability
- systematic exploration
- identification of root causes
- LLMs to support exploration of variants’ space
- incremental build of configuration space (Randrianaina et al. ICSE’22)
- debloating variability (Ternava et al. SAC’23)
- feature subset selection (Martin et al. SPLC’23)

Essentially, we want to reduce the dimensionality of the problem as well as the computational and human cost to foster verification of results and innovation.

Replicability is the holy grail!

- explore variants for robustness, validation, optimization and knowledge finding

Frictionless reproducibility: code+data+metrics

Deep variability is a problem (frictions!)
- evidence in many scientific domains

Deep variability is a solution (exploration!)
- fixing variability once and for all is not

Deep variability is a problem (frictions!)
- evidence in many scientific domains

Deep variability is a solution (exploration!)
- fixing variability once and for all is not
Backup slides (disclaimer: don’t try to understand everything ;))
What can we do? (robustness)

Robustness (trustworthiness) of scientific results to sources of variability

I have shown many examples of sources of variations and non-robust results...

Robustness should be rigorously defined (hint: it’s not the definition as given in computer science)

How to verify the effect of sources of variations on the robustness of given conclusions?

- actionable metrics?
- methodology? (eg when to stop?)
- variability can actually be leveraged to augment confidence
Variability in the analysis of a single neuroimaging dataset by many teams

Rotem Botvinik-Netzer, Felix Holzmeister, ... Tom Schönberg

Nature 582, 84–88 (2020) | Site this article
42k Accesses | 203 Citations | 2056 Altmetric | Metrics

Increasing Transparency Through a Multiverse Analysis

Sara Steegen¹ ⁹, Francis Tuerlinckx², Andrew Gelman², Wolf Vanpaemel³

Affiliations: expand
PMID: 27684465  DOI: 10.1177/1745691616658637

Many Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results

R. Sibertzahn, E. L. Uhlmann, D. P. Martin, more...

First Published August 23, 2018 | Research Article
https://doi.org/10.1177/2515245917747646

different data

different methods

different analyses

different assumptions

different variability

different software

different data
Deep software variability is...

a threat for reproducible research

“Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.”

an opportunity for replication

“A study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new analyses.”

“A study that refutes some scientific findings of another study, through the collection of new data (possibly with different methods) and completion of new analyses.”

robustifying and augmenting scientific knowledge
Reproducible Science as a Testing Problem

#1 Test Generation Problem (input)
   inputs: computing environment, parameters of an algorithm, versions of a library or tool, choice of a programming language

#2 Oracle Problem (output)
   we usually ignore the outcome! (open problems; open questions; new knowledge)

Input: System under Study (replicable)
Output: (scientific result)

“**Reproduction** of a computational study means running the same computation on the same input data, and then checking if the results are the same, or at least “close enough” when it comes to numerical approximations. Reproduction can be considered as **software testing** at the level of a complete study.”

**We don’t “test” in one run, in one computing environment, with one kind of input data, etc.**

“**Replication** of a scientific study (computational or other) means repeating a published protocol, respecting its spirit and intentions but **varying the technical details**. For computational work, this would mean using different software, running a simulation from different initial conditions, etc. The idea is to change something that everyone believes shouldn’t matter, and see if the scientific conclusions are affected or not.”

**It is the most interesting direction, basically for synthesizing new scientific knowledge!**

In both cases, there is the need to harness the combinatorial explosion of deep software variability.
Reproducible Science and Software Engineering
@acherm

aka Deep Software Variability for Replicability in Computational Science

Deep Questions?

Reproducibility and Replicability

Reproducible: Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.

Replication: A study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new analyses.

"Terminologies for Reproducible Research", Lorena A. Baresi, 2018

The ClearCross/Dycode/Endeavor terminology is broadly disseminated across disciplines (see Table 2). But the most adoption of an opposing terminology by two large professional groups—ACM and FASEB—made standardization awkward. The ACM published its rationale for adoption as based on the International Vocabulary of Metrology, but a close reading of the sources makes this justification tenuous. The source of the FASEB adoption is unclear, but there’s a chance that Caudron and Fager (2018) had an influence them. They, in turn, based their definitions on the emphatic but essentially flawed work of Drummond (2009).

Table 2: Grouping of terminologies, as in Table 1, by discipline.

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproducible</td>
<td>Authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results.</td>
</tr>
<tr>
<td>Replication</td>
<td>A study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new analyses.</td>
</tr>
</tbody>
</table>

Sometimes, variability is consistent/stable across layers and knowledge transfer is immediate.

But there are also interactions among variability layers and variability knowledge may not generalize.
Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code. (more)

Why does it matter?

Whilst anyone may inspect the source code of free and open source software for malicious flaws, most software is distributed pre-compiled with no method to confirm whether they correspond.

This incentivises attacks on developers who release software, not only via traditional exploitation, but also in the forms of political influence, blackmail or even threats of violence.

This is particularly a concern for developers collaborating on privacy or security software: attacking these typically result in compromising particularly politically-sensitive targets such as dissidents, journalists and whistleblowers, as well as anyone wishing to communicate securely under a repressive regime.

Whilst individual developers are a natural target, it additionally encourages attacks on build infrastructure as a successful attack would provide access to a large number of downstream computer systems. By modifying the generated binaries here instead of modifying the upstream source code, illicit changes are essentially invisible to its original authors and users alike.

The motivation behind the Reproducible Builds project is therefore to allow verification that no vulnerabilities or backdoors have been introduced during this compilation process. By promising identical results are always generated from a given source, this allows multiple third parties to come to a consensus on a “correct” result, highlighting any deviations as suspect and worthy of scrutiny.

This ability to notice if a developer or build system has been compromised then prevents such threats or attacks occurring in the first place, as any compromise can be quickly detected. As a result, front-liners cannot be threatened/coerced into exploiting or exposing their colleagues.

Several free software projects already, or will soon, provide reproducible builds.
“Linear model provides a good approximation of transformation between performance distributions of a system deployed in different hardware environments.”

what about variability of input data? compile-time options? version?
## Transfer Learning for Software Performance Analysis: An Exploratory Analysis
Jamshidi et al. ASE 2017

<table>
<thead>
<tr>
<th>SPEAR (SAT Solver)</th>
<th>X264 (video encoder)</th>
<th>SQLite (DB engine)</th>
<th>SaC (Compiler)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analysis time</strong></td>
<td><strong>Encoding time</strong></td>
<td><strong>Query time</strong></td>
<td><strong>Execution time</strong></td>
</tr>
<tr>
<td>14 options</td>
<td>16 options</td>
<td>14 options</td>
<td>50 options</td>
</tr>
<tr>
<td>16,384 configurations</td>
<td>4,000 configurations</td>
<td>1,000 configurations</td>
<td>71,267 configurations</td>
</tr>
<tr>
<td>SAT problems</td>
<td>Video quality/size</td>
<td>DB Queries</td>
<td></td>
</tr>
<tr>
<td>3 hardware</td>
<td>2 hardware</td>
<td>2 hardware</td>
<td></td>
</tr>
<tr>
<td>2 versions</td>
<td>3 versions</td>
<td>2 versions</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
ec_1 : [h_2 \rightarrow h_3, v_3, v_3] & \quad \text{SM} \quad 0.97 \\
ec_2 : [h_2 \rightarrow h_3, v_1, v_3] & \quad \text{S} \quad 0.96 \\
ec_3 : [h_1, w_1 \rightarrow w_2, v_3] & \quad \text{M} \quad 0.65 \\
ec_4 : [h_1, w_1 \rightarrow w_2, v_3] & \quad \text{M} \quad 0.67 \\
ec_5 : [h_1, w_3, v_2 \rightarrow v_3] & \quad \text{L} \quad 0.05 \\
ec_6 : [h_1, w_3, v_1 \rightarrow v_3] & \quad \text{L} \quad 0.06 \\
ec_7 : [h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3] & \quad \text{L} \quad 0.08 \\
ec_8 : [h_2 \rightarrow h_3, v_3, v_3, v_2 \rightarrow v_3] & \quad \text{VL} \quad 0.09
\end{align*}
\]

**Insight.** For non-severe hardware changes, we can linearly transfer performance models across environments.

**Insight.** The strength of the influence of configuration options is typically preserved across environments.

**Insight.** A large percentage of configurations are typically invalid in both source and target environments.
mixing deep variability: hard to assess the specific influence of each layer

very few hardware, version, and input data… but lots of runtime configurations (variants)

Let’s go deep with input data!
Practical impacts for users, developers, scientists, and self-adaptive systems

Threats to variability knowledge for performance property bitrate

- optimal configuration is specific to an input; a good configuration can be a bad one
- some options’ values have an opposite effect depending on the input
- effectiveness of sampling strategies (random, 2-wise, etc.) is input specific (somehow confirming Pereira et al. ICPE 2020)
- predicting, tuning, or understanding configurable systems without being aware of inputs can be inaccurate and… pointless
Computational science depends on software and its engineering

multi-million line of code base
multi-dependencies
multi-systems
multi-layer
multi-version
multi-person
multi-variant

from a set of scripts to automate the deployment to... a comprehensive system containing several features that help researchers exploring various hypotheses
x264 video encoder (compilation/build)

```
~ x264 --bframes 1 --ref 3 --cabac DiverSE-meeting.mp4 -o meeting13.webm
```

```
---
mathieuacher localhost.localdomain ..\x264-SLIMFAST\x264 x264-gcov □ .\configure --help
```

```
--disable-thread disable multithreaded encoding
--disable-win32threads disable win32threads (windows only)
--disable-interlaced disable interlaced encoding support
--bit-depth=BIT_DEPTH set output bit depth (8, 10, all) [all]
--chroma-format=FORMAT output chroma format (400, 420, 422, 444, all) [all]

Advanced options:
--disable-asmdisable platform-specific assembly optimizations
--enable-lto enable link-time optimization
--enable-debug add -g
--enable-gprof add -pg
--enable-strip build position-independent code

Cross-compilation:
--host=HOST build programs to run on HOST
--cross-prefix=PREFIX use PREFIX for compilation tools
--sysroot=sysroot root of cross-build tree

External library support:
--disable-avsdisable avisynth support
--disable-swsscale disable swscale support
--disable-lavf disable libavformat support
--disable-ffms disable ffmpegsource support
--disable-gpsd disable gpac support
--disable-lsmash disable lsmash support
```

**compile-time options**
What can we do? (#1 studies)

Empirical studies about deep software variability

- more subject systems
- more variability layers, including interactions
- more quantitative (e.g., performance) properties

with challenges for gathering measurements data:

- how to scale experiments? Variant space is huge!
- how to fix/isolate some layers? (eg hardware)
- how to measure in a reliable way?

Expected outcomes:

- significance of deep software variability in the wild
- identification of stable layers: sources of variability that should not affect the conclusion and that can be eliminated/forgotten
- identification/quantification of sensitive layers and interactions that matter
  - variability knowledge
What can we do? (#2 cost)

Reducing the cost of exploring the variability spaces

Many directions here (references at the end of the slides):

- **learning**
  - many algorithms/techniques with tradeoffs interpretability/accuracy
  - transfer learning (instead of learning from scratch)
- **sampling strategies**
  - uniform random sampling? t-wise? distance-based? …
  - sample of hardware? input data?
- **incremental build of configurations**
- **white-box approaches**
- …
Key results (for x264)

**Worth tuning software at compile-time:** gain about 10% of execution time with the tuning of compile-time options (compared to the default compile-time configuration). The improvements can be larger for some inputs and some runtime configurations.

**Stability of variability knowledge:** For all the execution time distributions of x264 and all the input videos, the worst correlation is greater than 0.97. If the compile-time options change the scale of the distribution, they do not change the rankings of run-time configurations (i.e., they do not truly interact with the run-time options).

Reuse of configuration knowledge:

\[ f_1 = \beta \times f_2 + \alpha \]

- Linear transformation among distributions
- Users can also trust the documentation of run-time options, consistent whatever the compile-time configuration is.

Embracing Deep Variability For Reproducibility & Replicability

Mathieu Acher
Univ Rennes, Inria, CNRS, IRISA, IUF
Rennes, France

Georges Auren Ramdhanina
Univ Rennes, Inria, CNRS, IRISA
Rennes, France

Benoit Combemale
Univ Rennes, Inria, CNRS, IRISA, IUF
Rennes, France

Jean-Marc Jézéquel
Univ Rennes, Inria, CNRS, IRISA, IUF
Rennes, France

ABSTRACT

Reproducibility (ac.s, determination in some cases) constitutes a fundamental input in various fields of computer science, such as
hunting point computations in numerical analysis and simulation,
concurrency models in parallelism, irreducible builds for third parties integration and packaging, and containerization for
execution environments. These concepts, while pervasive across dif-
ferent concerns, often elude intimate interdependencies, making
it challenging to achieve a comprehensive understanding. In this
short and visionary paper, we define two applications of software
engineering techniques, specifically variability management, to
systematically identify and exploit points of variability that may
give rise to reproducibility issues (e.g., language, libraries, compile,
virtual machines, OS, environment variables, etc). The primary ob-
jectives are: (i) gaining insights into the variability layers and their
possible interactions, (ii) exploring and documenting configurations
for the sake of reproducibility, and (iii) exploring diverse configu-
ration settings to replicate, and hence validate and ensure the robustness
of results. By adopting these methodologies, we aim to address the
complexities associated with reproducibility and replicability in
modern software systems and environments, facilitating a more comprehensive and nuanced perspective on these critical aspects.

1 INTRODUCTION

Many scientific domains need to process large amount of data with
more and more complex computations. For instance, studies about
climate modeling and change involve the design of mathematical
models, the mining and analysis of data, the creation of large sim-
ulations, etc. [10, 14, 15]. These computational tasks rely on various
kinds of software, from a set of scripts to automate the deployment
to a comprehensive system containing several features that help
researchers exploring various hypotheses. It is not an overstate-
ment to say that computational science depends on software and its
engineering [2, 14, 56].

Our field of study is computer science that is a result obtained by
an experimental (e.g., a simulation) or be achieved again with a high-
degree of agreement. But despite the availability of data and code,
several studies report that the same data analyzed with different
software can lead to different results [6, 9, 13, 15, 22, 31, 41, 42, 55]. For instance, applications of different analysis pipelines, alterations

⇒ We aim to address the complexities associated with reproducibility and replicability in modern software systems and environments, facilitating a more comprehensive and nuanced perspective on these critical “factors”.

https://hal.science/hal-04582287
Multi-Level Analysis of Compiler-Induced Variability and 
Performance Tradeoffs

Michael Bentley
Ian Briggs
Ganesh Gopalakrishnan
mbentley@cs.utah.edu
ianbriggsutah@gmail.com
ganesh@cs.utah.edu
University of Utah

Dong H. Ahn
Ignacio Laguna
Gregory L. Lee
Holger E. Jones
ahn1@llnl.gov
lagunaperalt1@llnl.gov
lee18@llnl.gov
jones19@llnl.gov
Lawrence Livermore National Laboratory

**Definition of Reproducibility.** Given the growing heterogeneity of hardware and software, one cannot always define reproducibility as achieving bitwise reproducible results. Instead, we view a reproducible computation as one that produces a result within an “acceptable range” of a trusted baseline answer. In FLiT, we rely on the application developer to provide an acceptance testing function that (indirectly) defines this range.

---

Table 1: Compilers used in the MFEM study with summary statistics. The best flags are chosen by the best average speedup across all MFEM examples. The average speedup over all 19 MFEM examples is reported and is calculated relative to the speed of g++ `-O2`.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Released</th>
<th># Variable Runs</th>
<th>Best Flags</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc-8.2.0</td>
<td>26 July 2018</td>
<td>78 of 1,228 (6.0%)</td>
<td><code>-O2 -fun-safe-math-optimizations</code></td>
<td>1.097</td>
</tr>
<tr>
<td>clang-6.0.1</td>
<td>05 July 2018</td>
<td>24 of 1,368 (1.8%)</td>
<td><code>-O3 -fun-safe-math-optimizations</code></td>
<td>1.042</td>
</tr>
<tr>
<td>icpc-18.0.3</td>
<td>16 May 2018</td>
<td>984 of 1,976 (49.8%)</td>
<td><code>-O2 -fp-model fast=2</code></td>
<td>1.056</td>
</tr>
</tbody>
</table>

---

Figure 1: Multi-level workflow. Levels are (1) determine variability-inducing compilations, (2) analyze the space of reproducibility and performance, and (3) debug variability by identifying files and functions causing variability.
exec (software) = exec_repro (software)

or

exec(software) \sim= exec(software_repro)

(difference: exec_repro is another execution environment... and so somehow differs or not with exec; or we consider that software differs...)

(exec: execution? what's the outcome then? in fact execution can be replaced by "build"... which is another kind of execution)

exec (software) ?= exec_repro (software)

software \sim= software_repro

exec (software, hardware)

exec (software, hardware, compiler, input_data, operating_system, bios, container, hypervisor, dependencies_versions)

exec (v1, v2, ..., vN) \sim= exec_repro (v1', v2', ..., vN')

for i in [1, n], v_{i} \sim= v_{i}' (or not!)

\sim= is specific to a domain, to a usage, etc.

\sim= can be over the N layers or over N' layers (N' < N)

\sim= can be specific to some pairs elements (eg we know that with this hardware, the exec time is multiplied by 2)

for instance, we know the \sim= between a software configuration with any hardware (but if the compiler changes, then the \sim= should be "tuned" accordingly)

also \sim= can be defined between a configuration set and an hardware set (eg performance distribution)
Exact same results? No

Definition of Reproducibility. Given the growing heterogeneity of hardware and software, one cannot always define reproducibility as achieving bitwise reproducible results. Instead, we view a reproducible computation as one that produces a result within an "acceptable range" of a trusted baseline answer. In FLiT, we rely on the application developer to provide an acceptance testing function that (indirectly) defines this range.

Figure 1: Multi-level workflow. Levels are (1) determine variability-inducing compilations, (2) analyze the space of reproducibility and performance, and (3) debug variability by identifying files and functions causing variability.
Frictionless reproducibility (annotated bibliography; grey literature)

https://hdsr.mitpress.mit.edu/pub/8dggwqiu/release/1 The Mechanics of Frictionless Reproducibility, B Recht

interesting historical perspective on research in neural networks (NeurlPs 87 titles are shockingly still relevant); really love some parts about random experiments, science as a “massively parallel genetic algorithm” or the discussions on the difficulty of using ML/DL software (completely aligned with my terrible experience of Weka GUI in ~2006)

https://www.argmin.net/p/the-department-of-frictionless-reproducibility

Progress and frictionless reproducibility

Inspired by Thomas Kuhn (1962), we can think of the scientific and engineering process as a massively parallel genetic algorithm. If we want to improve upon the systems we currently have, we might try a small perturbation to see if we get an improvement. If we can find a small change that improves some desired outcome, we could change our systems to reflect this improvement. If we continually search for these improvements and work hard to demonstrate their value, we may head in a better direction over time.

For scientific endeavors, we could perhaps gauge ‘better’ or ‘worse’ by performing random experiments—not randomized experiments per se, but random experiments in the sense of trying potentially surprising improvements. If our small tweak results in better outcomes, we can attempt to convince a journal editor or conference program committee to publish it. And this communication gives everyone else a new starting point for their own random experimentation.

A single investigator can only make so much progress by random searching alone, but random search is pleasantly parallelizable. Competing scientists can independently try their own random ideas and publish their results. Sometimes an individual result is so promising that the herd of experimenters all flock around the good idea, hoping to strike gold on a nearby improvement and bring home bragging rights. To some, this looks like an inefficient mess. To others, it looks like science.

https://hdsr.mitpress.mit.edu/pub/8dqgwqiu/release/1 The Mechanics of Frictionless Reproducibility, B Recht
Data sharing and frictions

“Data set benchmarking and competitive testing took over machine learning in the late 1980s. Email and file transfer were becoming more accessible. The current specification of FTP was finalized in 1985. In 1987, a PhD student at UC Irvine named David Aha put up an FTP server to host data sets for empirically testing machine learning methods. Aha was motivated by service to the community, but he also wanted to show his nearest-neighbor methods would outperform Ross Quinlan’s decision tree induction algorithms. He formatted his data sets using the ‘attribute-value’ representation that a rival researcher, Ross Quinlan (1986), had used. And, so, the UC Irvine Machine Learning Repository was born.”

“Improvements in computing greased the wheels, giving us faster computers, faster data transfer, and smaller storage footprints. But computing technology alone was not sufficient to drive progress. Friendly competition with Quinlan inspired Aha to build the UCI repository. And more explicit competitions were also crucial components of the success.”

The Mechanics of Frictionless Reproducibility, B Recht, 2024

https://hdsr.mitpress.mit.edu/pub/8dqgwqiui/release/1
Stas Bekman (@StasBekman) Jan 20

Floating point math discrepancies with some pretrained LM models can be an issue.

I was debugging today a weird discrepancy between Llama-2-7b inference results which proved to be triggered by whether `from_pretrained` was called...

This is from the mps device:
Figure 2: Feature model (excerpt). Inverse (resp. Relation-WithPi) corresponds to checking the property \((x \cdot z)/(y \cdot z) = x/y\) (resp. \((x \cdot z \cdot \pi)/(y \cdot z \cdot \pi) = x/y\)) with \(z, y \neq 0\)

def equality_test(equality_check: EqualityCheck, x: y: z) -> bool:
    if equality_check == EqualityCheck.ASSOCIATIVITY:
        return x/(y+z) == (x+y)*z
    elif equality_check == EqualityCheck.MULT_INV:
        return (x * z) / (y * z) == x / y
    elif equality_check == EqualityCheck.MULT_INV_PI:
        return (x * z * math.pi) / (y * z * math.pi) == (x / y)
Figure 2: Feature model (excerpt). Inverse (resp. RelationWithPi) corresponds to checking the property \((x * z) / (y * z) = x / y\) (resp. \((x * z * \pi) / (y * z * \pi) = x / y\)) with \(z, y \neq 0\)

```rust
if error_margin:
    variability_misc = f"--error_margin {error_margin}"
    cmd_args = ['--error_margin', error_margin]
    cmd_args = ['cargo',
        'run',
        '..-features',
        feature,
        '..',
        '..',
        '..-error_margin',
        error_margin,
    ]

fn associativity_test(config: &Config) -> bool {
    let mut rng = thread_rng();
    // TODO: this variant for generating random
    let x = rng.gen::<f64>();
    let y = rng.gen::<f64>();
    let z = rng.gen::<f64>();
    let x = rng.gen_range(0.000_000_000_000_001..100.0); // TODO: variation point for range min, max value
    let y = rng.gen_range(0.000_000_000_000_001..100.0);
    let z = rng.gen_range(0.000_000_000_000_001..100.0);

    check_ratio(config, x, y, z)
}

fn proportion(config: &Config, number: i32, seed_val: u64) -> i32 {
    let mut ok = 0;
    for _ in 0..number {
        if associativity_test(config) {
            ok += 1;
        }
    }
    ok * 100 / number
}
```
<table>
<thead>
<tr>
<th>Language</th>
<th>Library</th>
<th>System</th>
<th>Compiler</th>
<th>VariabilityMisc</th>
<th>EqualityCheck</th>
<th>NumberGenerations</th>
<th>Repeat</th>
<th>min</th>
<th>max</th>
<th>std</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>100.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Perl</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>60.0</td>
<td>71.0</td>
<td>6.0</td>
<td>0.0</td>
<td>65.1</td>
</tr>
<tr>
<td>Perl</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>51.0</td>
<td>63.0</td>
<td>6.0</td>
<td>0.0</td>
<td>55.9</td>
</tr>
<tr>
<td>Perl</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>100.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Perl</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>62.0</td>
<td>62.0</td>
<td>0.0</td>
<td>0.0</td>
<td>62.0</td>
</tr>
<tr>
<td>Perl</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>47.0</td>
<td>47.0</td>
<td>0.0</td>
<td>0.0</td>
<td>47.0</td>
</tr>
<tr>
<td>Go</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>associativity</td>
<td>100</td>
<td>10</td>
<td>71.0</td>
<td>82.0</td>
<td>6.0</td>
<td>0.0</td>
<td>76.0</td>
</tr>
<tr>
<td>Go</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>mult-inverse</td>
<td>100</td>
<td>10</td>
<td>58.0</td>
<td>78.0</td>
<td>6.0</td>
<td>0.0</td>
<td>66.0</td>
</tr>
<tr>
<td>Go</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>mult-inverse</td>
<td>100</td>
<td>10</td>
<td>42.0</td>
<td>55.0</td>
<td>7.0</td>
<td>0.0</td>
<td>43.4</td>
</tr>
<tr>
<td>Go</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>associativity</td>
<td>100</td>
<td>10</td>
<td>81.0</td>
<td>81.0</td>
<td>0.0</td>
<td>0.0</td>
<td>81.0</td>
</tr>
<tr>
<td>Go</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>mult-inverse</td>
<td>100</td>
<td>10</td>
<td>70.0</td>
<td>70.0</td>
<td>0.0</td>
<td>0.0</td>
<td>70.0</td>
</tr>
<tr>
<td>Go</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>mult-inverse</td>
<td>100</td>
<td>10</td>
<td>56.0</td>
<td>56.0</td>
<td>0.0</td>
<td>0.0</td>
<td>56.0</td>
</tr>
<tr>
<td>R</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>100.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>R</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>62.0</td>
<td>72.0</td>
<td>6.0</td>
<td>0.0</td>
<td>66.0</td>
</tr>
<tr>
<td>R</td>
<td>seed None</td>
<td>None</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>47.0</td>
<td>57.0</td>
<td>6.0</td>
<td>0.0</td>
<td>53.1</td>
</tr>
<tr>
<td>R</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>100.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>R</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>67.0</td>
<td>67.0</td>
<td>0.0</td>
<td>0.0</td>
<td>67.0</td>
</tr>
<tr>
<td>R</td>
<td>seed None</td>
<td>None</td>
<td>seed 42</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>53.0</td>
<td>53.0</td>
<td>0.0</td>
<td>0.0</td>
<td>53.0</td>
</tr>
<tr>
<td>Julia</td>
<td>seed None</td>
<td>strict-equality</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>74.0</td>
<td>90.0</td>
<td>6.0</td>
<td>0.0</td>
<td>82.5</td>
</tr>
<tr>
<td>Julia</td>
<td>seed None</td>
<td>strict-equality</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>60.0</td>
<td>79.0</td>
<td>7.0</td>
<td>0.0</td>
<td>68.6</td>
</tr>
<tr>
<td>Julia</td>
<td>seed None</td>
<td>strict-equality</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>49.0</td>
<td>59.0</td>
<td>6.0</td>
<td>0.0</td>
<td>54.3</td>
</tr>
<tr>
<td>Julia</td>
<td>seed None</td>
<td>strict-equality</td>
<td>seed None</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>89.0</td>
<td>89.0</td>
<td>0.0</td>
<td>0.0</td>
<td>89.0</td>
</tr>
<tr>
<td>Julia</td>
<td>seed 42 strict-equality</td>
<td>seed None</td>
<td>seed 42 strict-equality</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>73.0</td>
<td>73.0</td>
<td>0.0</td>
<td>0.0</td>
<td>73.0</td>
</tr>
<tr>
<td>Julia</td>
<td>seed 42 strict-equality</td>
<td>seed None</td>
<td>seed 42 strict-equality</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>55.0</td>
<td>55.0</td>
<td>0.0</td>
<td>0.0</td>
<td>55.0</td>
</tr>
<tr>
<td>Julia</td>
<td>seed None approximate equality of julia lang</td>
<td>seed None</td>
<td>seed None approximate equality of julia lang</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>100.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Julia</td>
<td>seed None approximate equality of julia lang</td>
<td>seed None</td>
<td>seed None approximate equality of julia lang</td>
<td>ASSOCIATIVITY</td>
<td>100</td>
<td>10</td>
<td>100.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

https://github.com/FAMILIAR-project/reproducibility-associativity/