



HAL
open science

Deep Software Variability and Frictionless Reproducibility

Mathieu Acher

► **To cite this version:**

Mathieu Acher. Deep Software Variability and Frictionless Reproducibility. 2024, pp.1-121. hal-04601752

HAL Id: hal-04601752

<https://hal.science/hal-04601752v1>

Submitted on 5 Jun 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Public Domain

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.

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 Tërnavá, Olivier Barais, and the
whole DiverSE and RIPOST teams

*random order, incomplete

AGENDA

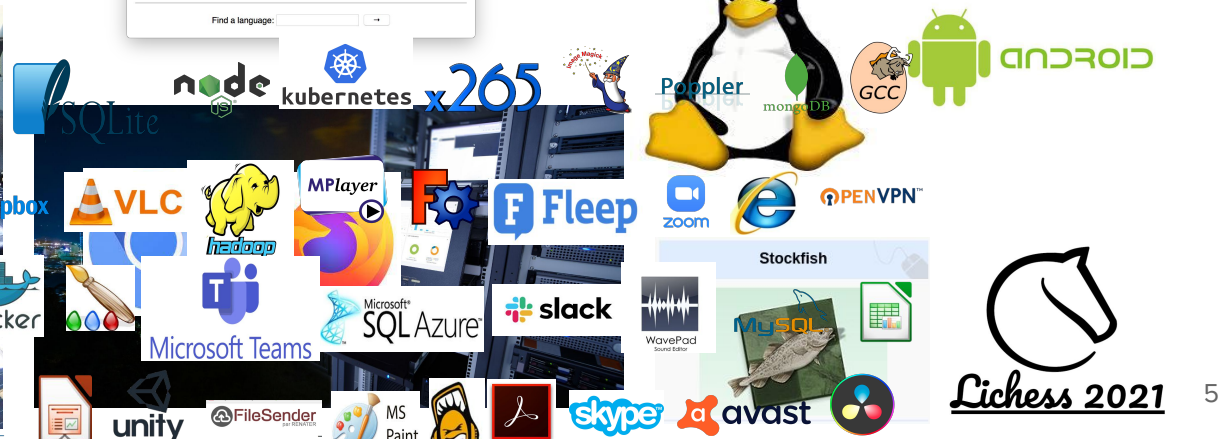
Frictionless Reproducibility and (Deep) Software (Variability)

Problem: Variability and Frictions

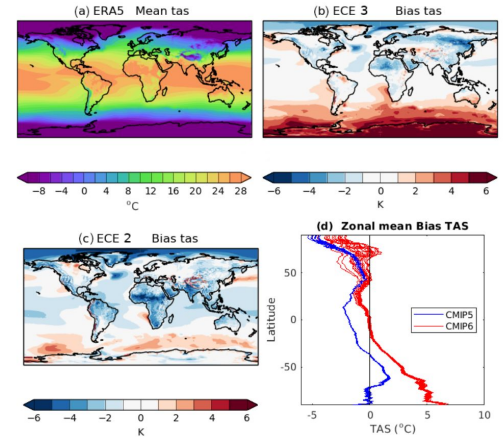
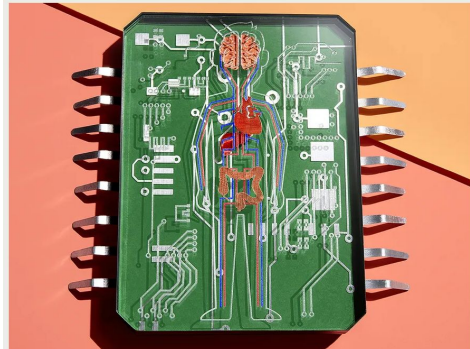
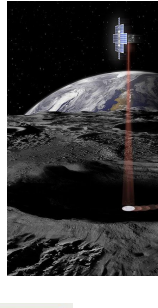
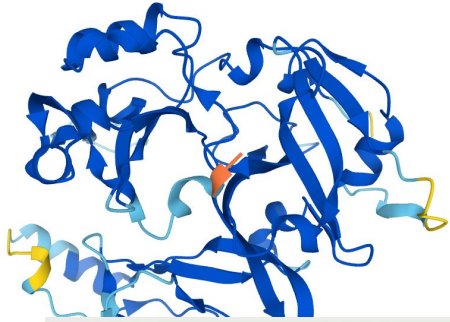
Solution: Variability and Exploration

Discussions

SOFTWARE VARIANTS ARE EATING THE WORLD



Science is changing: Computation-based research

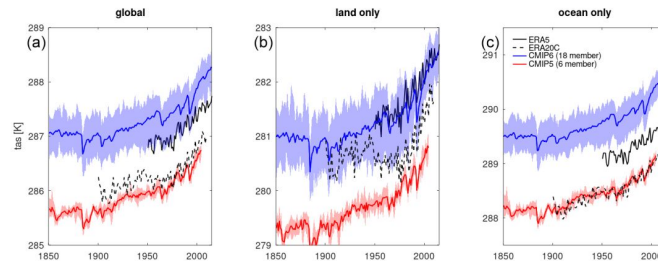
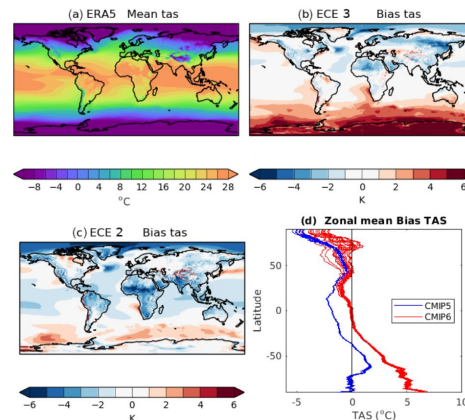


Computational science depends on software and its engineering



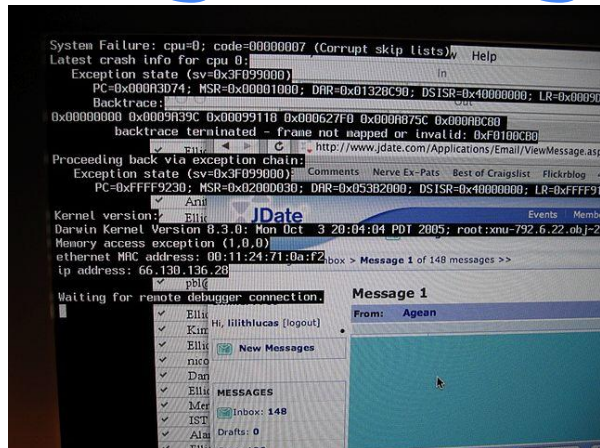
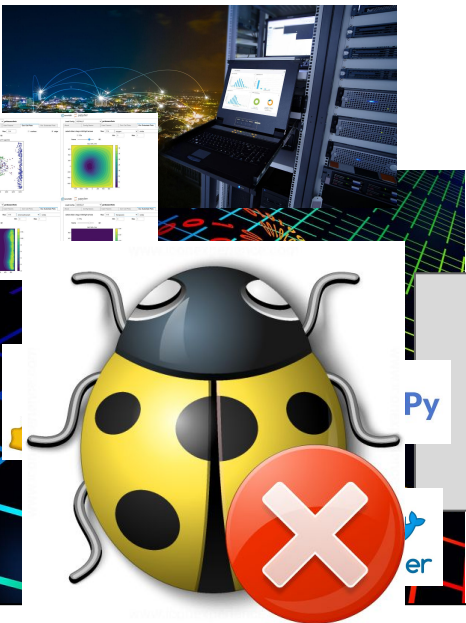
design of mathematical model
mining and analysis of data
executions of large simulations
problem solving
executable paper

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

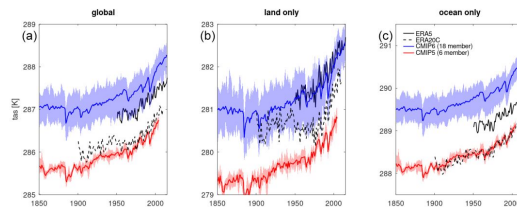
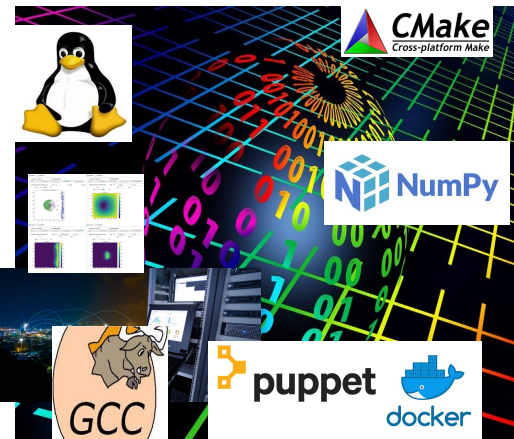
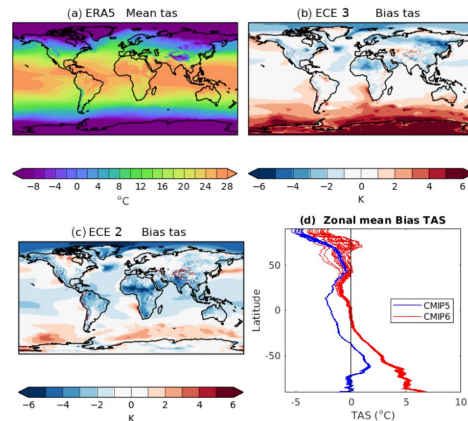
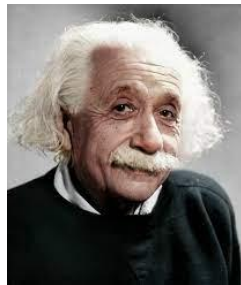


```
In [6]: sess.run(tf.svd(tf_matrix))
Out[6]: (array([[ 9.99999987e-01,  1.48747023e-03,  4.88133628e-06,
  4.69611084e-06,  4.37980998e-06,  3.45290823e-06,
  1.14686304e-06,  3.10980795e-06,  2.97625912e-06,
  2.65099743e-06,  1.91537106e-06,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00], dtype=float32),
array([[ 1.00000000e+00,  9.82503479e-05,  -2.52892733e-06,
  6.43756945e-07,  -2.61291496e-06,  -3.18651830e-06,
  0.00000000e+00,  nan,  nan,  nan,
  8.60062854e-10,  1.93595340e-09,  0.00000000e+00,
  -1.24836730e-09,  3.83645737e-09,  -3.90316446e-09,
  nan,  -7.00323994e-07],
[ -7.27595761e-11,  7.15255737e-07,  -2.68207733e-02,
  -6.68754578e-01,  1.68675050e-01,  -2.37232931e-02,
```

Dealing with software collapse: software stops working eventually
Konrad Hinsin 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”

<http://throwgrammarfromthetrain.blogspot.com/2010/10/definition-of-insanity.html>

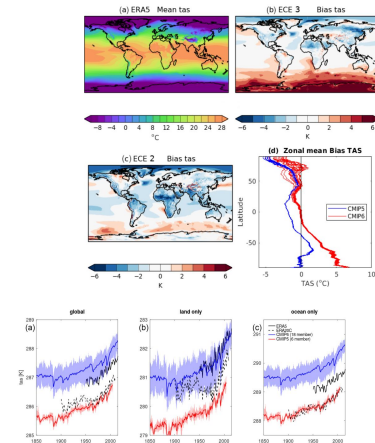
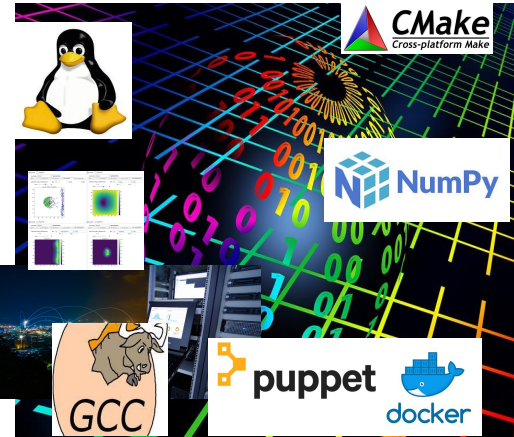


Reproducibility

“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)

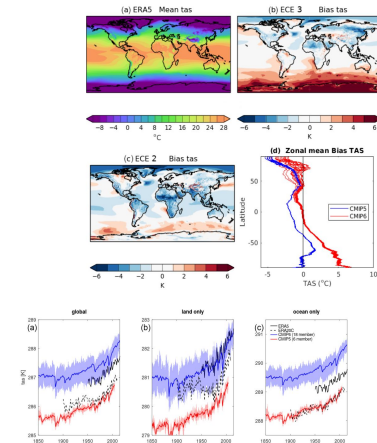
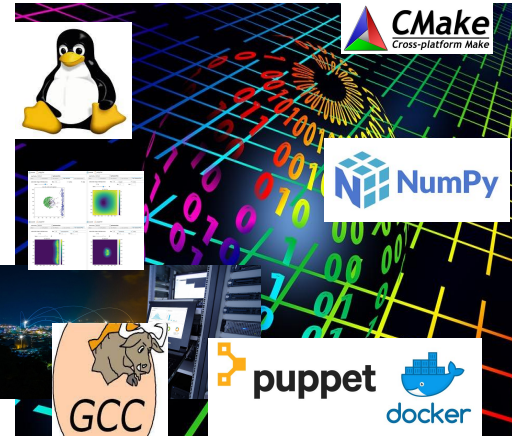


Reproducibility and Replicability

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

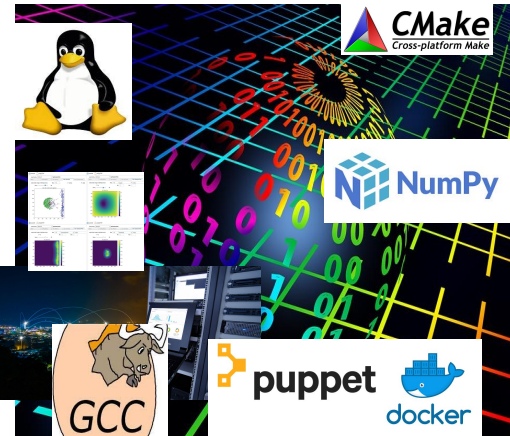


Reproducibility and Replicability

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



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.

A	B1	B2
political science economics	signal processing scientific computing econometry epidemiology clinical studies internal medicine physiology (neuro) computational biology biomedical research statistics	microbiology, immunology (FASEB) computer science (ACM)

*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 artifact review and badging initiative. ACM took action to update all prior badging to ensure consistency.

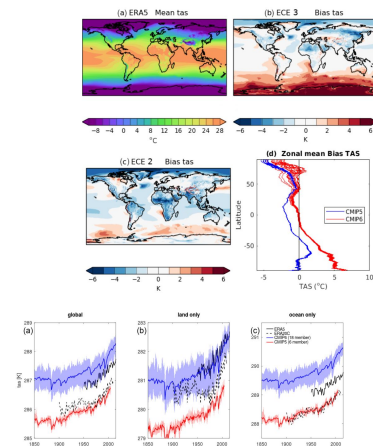
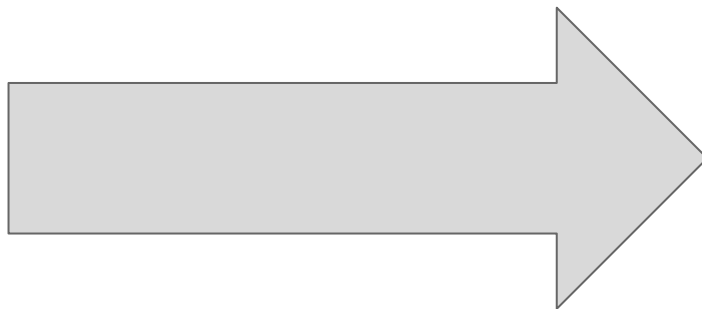
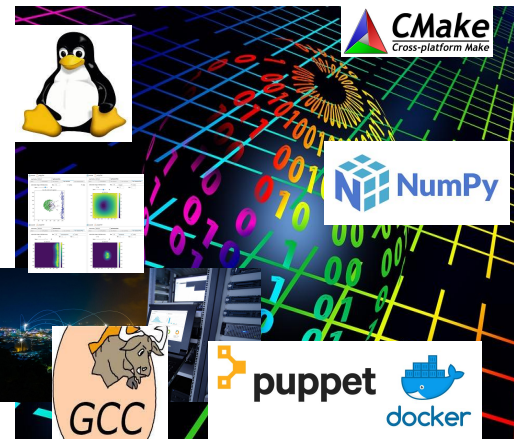
Reproducibility and Replicability

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

Results Reproducibility: A result is reproducible if a reimplementaion 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.

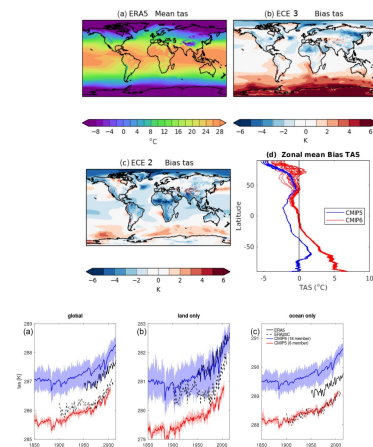
“Unreproducible Research is Reproducible”, Bouthillier et al., ICML 2019



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.

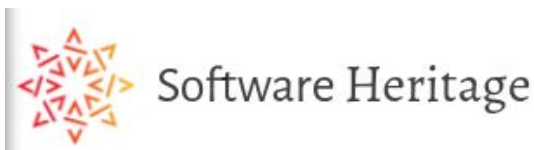
with many challenges related to data acquisition, knowledge organization/sharing, etc.

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 theories in the long run.

<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/>

OpenReview.net



GitHub
Enterprise



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.



CALL FOR CHALLENGE CASES

[Home / Call for Papers / Call for challenge cases](#)

ARTIFACT EVALUATION

Authors of accepted research papers are invited to submit the artifacts associated with their paper for evaluation. To do so, they should submit a PDF via EasyChair (select the R Artifacts track). The PDF should contain a stable URL (or DOI) to the artifacts. The PDF should contain the steps or general instructions to execute/analyze the artifact. Each artifact submission will be reviewed by at least two reviewers.

According to ACM's "Result and Artifact Review and Badging" policy, an "artifact" is "a digital object that was either created by the authors to be used as part of the study or generated by the experiment itself [...] which can include] software systems, scripts used to run experiments, input datasets, raw data collected in the experiment, or scripts used to analyze results."



The idea of the challenge track is to provide participants with a set of case studies that tackle relevant problems and challenge the state of the art. The challenge track happens in two phases. In the first phase, there will be a call for cases. Submitted cases will be reviewed by the challenge co-chairs to ensure that the information is clearly described. Accepted cases will be part of the official conference proceedings.

In this track, an industrial performance dataset will be provided. The participants are invited to come up with research questions about the dataset, and study those. The challenge is open-ended: participants can choose the research questions that they find most interesting. The proposed approaches and/or tools and their findings are discussed in short papers, and presented in the main conference.

ICPE 2022

13th ACM/SPEC International Conference on Performance Engineering

Lamb and Zacchiroli “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022

<https://arxiv.org/pdf/2104.06020>

(best paper award IEEE Software for year 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.”



Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code. ([more](#))

Frictionless reproducibility

Statistics > Other Statistics

[Submitted on 2 Oct 2023]

Data Science at the Singularity

David Donoho

<https://arxiv.org/abs/2310.00865>

<https://hdsr.mitpress.mit.edu/pub/g9mau4m0/release/2>

“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

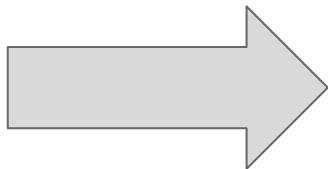
[FR-1: Data] “Datafication of everything, with a culture of research data sharing.”

[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.”



CODE



performance
metric



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]



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

Table 1: Leave-One-outs, and what is blocked

Frictionless reproducibility

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

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

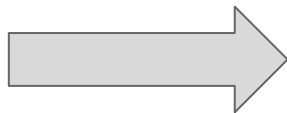
frictionless reproducibility = [FR-1: Data] + [FR-2: Re-execution] + [FR-3: Challenges]

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

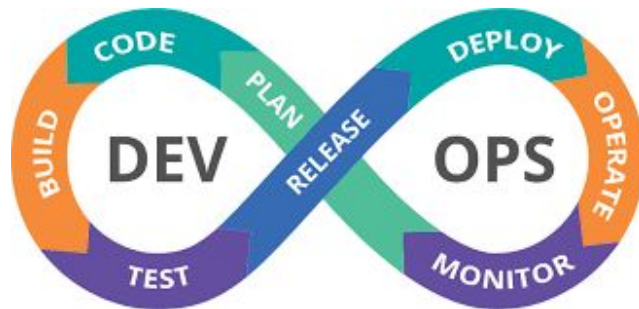
On the other hand, we know that the power of a simple scoring function is dangerous (e.g., Goodhart's law)

“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)



performance
metric



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.

```
1 diff --git a/core/defenses/sabre.py b/core/defenses/sabre.py
2 index fe509e6..bf13629 100644
3 --- a/core/defenses/sabre.py
4 +++ b/core/defenses/sabre.py
5 @@ -165,7 +165,7 @@ class SabreWrapper(nn.Module):
6     model = Sabre(eps=eps, wave=wave, use_rand=use_rand, n_variants=n_variants)
7     self.core = model
8     self.base_model = base_model
9 -     self.transform = BPDWrapper(lambda x, lambda_r: model.transform(x, lambda_r).float())
10 +     self.transform = (lambda x, lambda_r: model.transform(x, lambda_r).float())
11 @property
12 def lambda_r(self):
```

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)

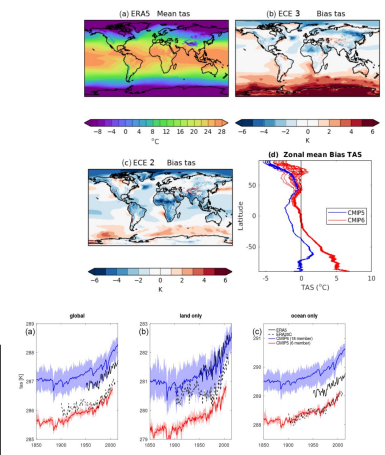
Reproducible science... with frictions

“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.



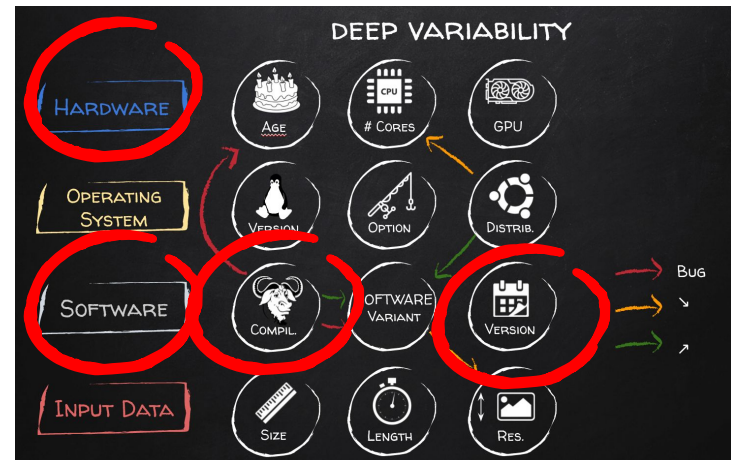
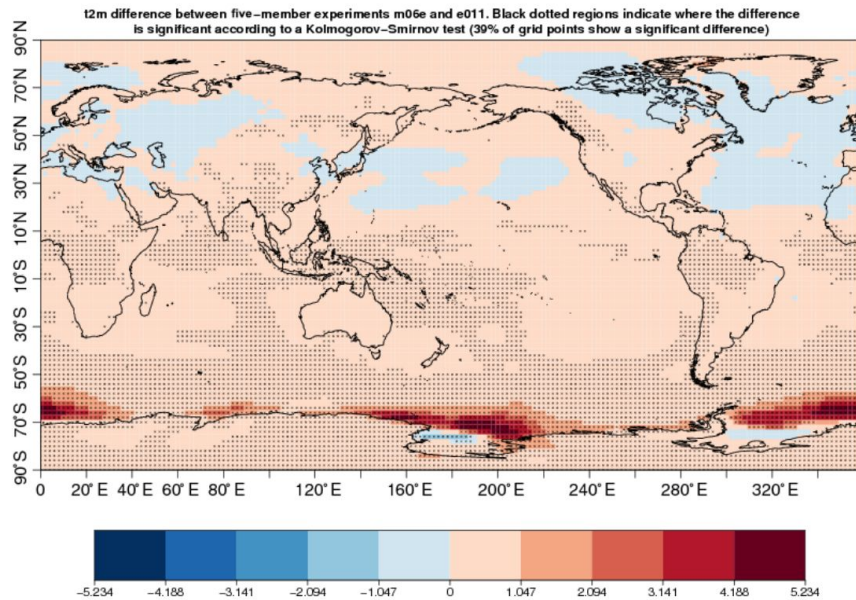
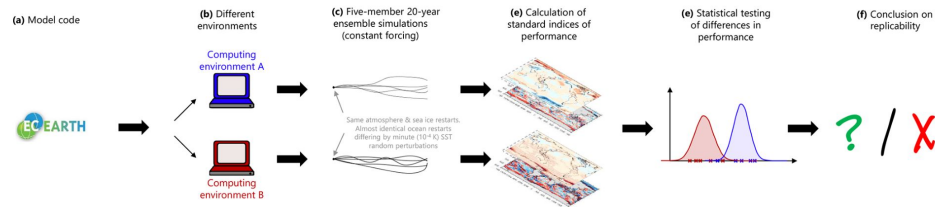
from a set of scripts to automate the deployment to... a comprehensive system containing several features that help researchers exploring various hypotheses



Replicability of the EC-Earth3 Earth system model under a change in computing environment

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

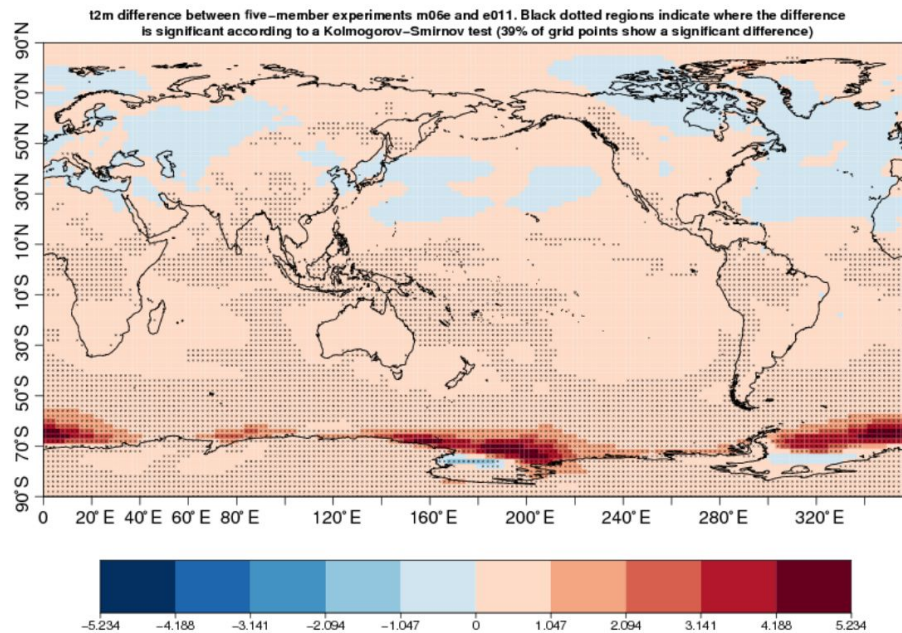
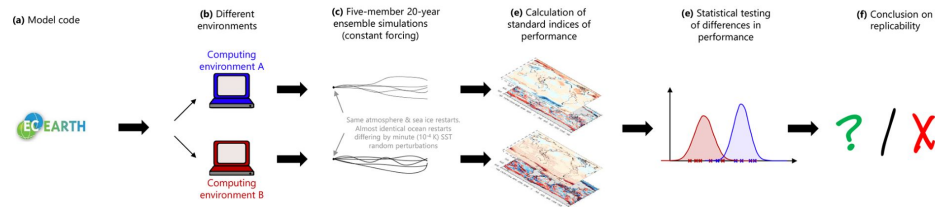
Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one “non-replicable” case (see below) and one replicable case.



Replicability of the EC-Earth3 Earth system model under a change in computing environment

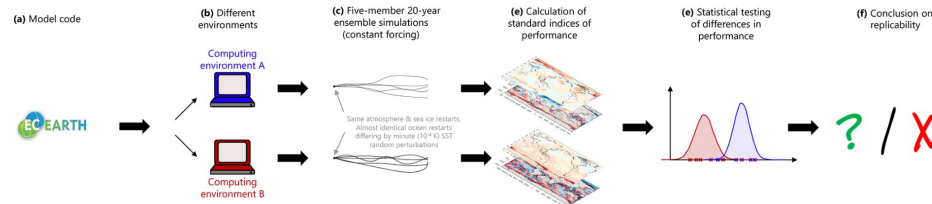
François Massonnet^{1,2}, Martin Ménégoz^{2,3}, Mario Acosta², Xavier Yepes-Arbós², Eleftheria Exarchou², and Francisco J. Doblas-Reyes^{2,4}

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 Massonnet^{1,2}, Martin Ménégoz^{2,3}, Mario Acosta², Xavier Yepes-Arbós², Eleftheria Exarchou², and Francisco J. Doblas-Reyes^{2,4}



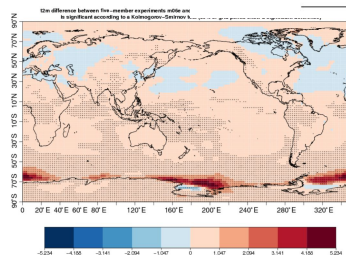
Can a coupled **ESM** simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one **“non-replicable”** case (see below) and one replicable case.

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

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

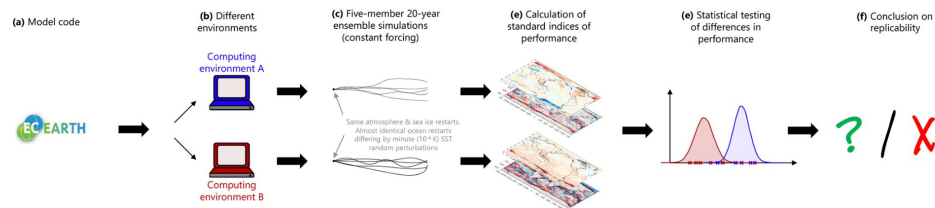
Table 2. The four experiments considered in this study.

Experiment ID	e011	m06e	a0gi	a0go
Computing environment	ECMWF-CCA	MareNostrum3	ECMWF-CCA	MareNostrum3
EC-Earth version	3.1	3.1	3.2	3.2
Processors (IFS+NEMO+OASIS)	598 (480 + 96 + 22)	512 (384 + 96 + 22)	432 (288 + 144)	416 (288 + 128)
F flags	-O2 -g -traceback -vec-report0 -r8 -vec-report0 -r8	-O2 -g -traceback -vec-report0 -r8 -vec-report0 -r8	-O2 -g -traceback -r8 -fp-model strict -fp-model strict -xHost	-O2 -fp-model precise -xHost -g -traceback -r8
C flags	-O2 -g -traceback	-O2 -g -traceback	-O2 -g -traceback -fp model strict -xHost	-O2 -fp-model precise -xHost -g -traceback
LD flags	-O2 -g -traceback	-O2 -g -traceback	-O2 -g -traceback -fp-model strict -xHost	-O2 -fp-model precise -xHost -g -traceback
Output size	141.8 GB	141.6 GB	101.3 GB	101.3 GB



Replicability of the EC-Earth3 Earth system model under a change in computing environment

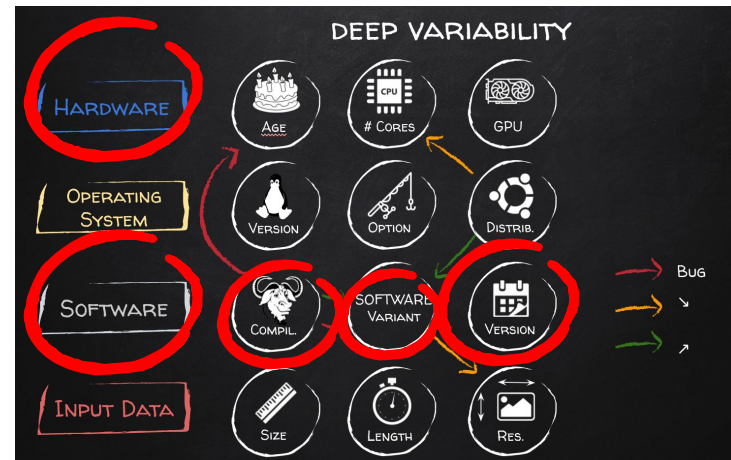
François Massonnet^{1,2}, Martin Ménégoz^{2,3}, Mario Acosta^{1,2}, Xavier Yepes-Arbós^{1,2}, Eleftheria Exarchou^{1,2}, and Francisco J. Doblas-Reyes^{1,2,4}



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?).

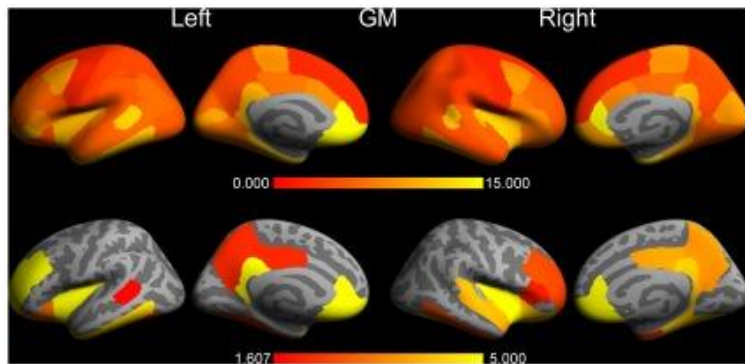


Should software version numbers determine science?

> [PLoS One](#). 2012;7(6):e38234. doi: 10.1371/journal.pone.0038234. Epub 2012 Jun 1.

The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements

Ed H B M Gronenschild ¹, Petra Habets, Heidi I L Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, Machteld Marcelis

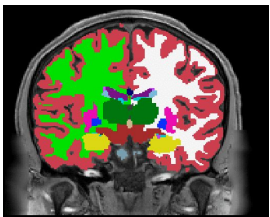


Significant differences were revealed between FreeSurfer version v5.0.0 and the two earlier versions. [...] About a factor two smaller differences were detected between Macintosh and Hewlett-Packard workstations and between OSX 10.5 and OSX 10.6. The observed differences are similar in magnitude as effect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

see also Krefting, D., Scheel, M., Freing, A., Specovius, S., Paul, F., and Brandt, A. (2011). "Reliability of quantitative neuroimage analysis using freesurfer in distributed environments," in *MICCAI Workshop on High-Performance and Distributed Computing for Medical Imaging*. (Toronto, ON).

“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



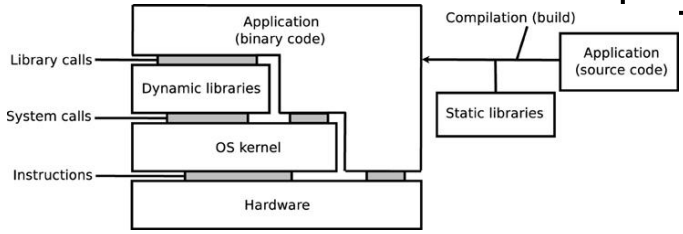
The implementation of mathematical functions manipulating single-precision **floating-point numbers in libmath** has evolved during the last years, leading to numerical differences in computational results. While these differences have little or no impact on simple analysis pipelines such as brain extraction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

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

“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

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**.

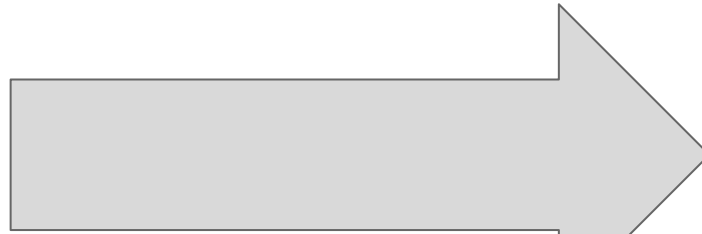


	Cluster A	Cluster B
Applications	Freesurfer 5.3.0, build 1 FSL 5.0.6, build 1 CIVET 1.1.12-UCSF, build 1	Freesurfer 5.3.0, build 1 and 2 FSL 5.0.6, build 1 and 2 CIVET 1.1.12-UCSF, build 1
Interpreters	Python 2.4.3, bash 3.2.25, Perl 5.8.8, tcsh 6.14.00	Python 2.7.5, bash 4.2.47, Perl 5.18.2, tcsh 6.18.01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

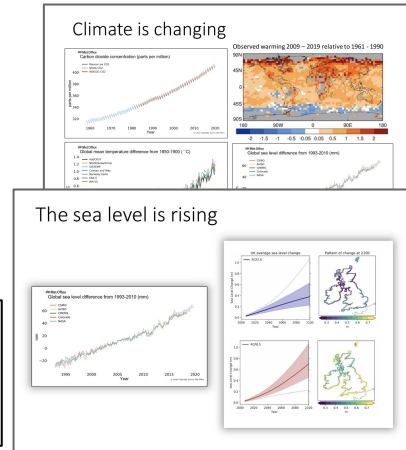
Reproducible science as a (deep) software variability problem

“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.



from a set of scripts to automate the deployment to... a comprehensive system containing several features that help researchers exploring various hypotheses



An aerial, black and white photograph of a city, likely New York City, showing a dense grid of buildings. Overlaid on the image are several orange rectangular labels with white text, each representing a different layer of variability in software. The labels are arranged vertically on the left side of the image, with a diagonal label 'version variability' on the right side of the image.

deep software variability

software application variability

input data variability

build variability

compiler variability

container variability

hypervisor variability

operating system variability

hardware variability

version variability

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 $(x+y)+z == x+(y+z)$?

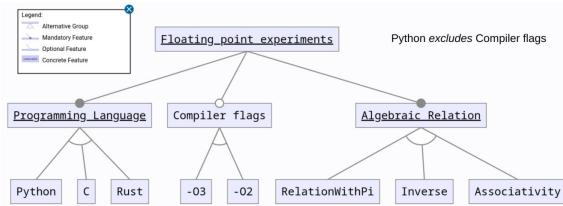


Figure 2: Feature model (excerpt). Inverse (resp. Relation-WithPi) corresponds to checking the property $(x * z) / (y * z) = x / y$ (resp. $(x * z * \pi) / (y * z * \pi) = x / y$) with $z, y \neq 0$



Parameters, Input Data	e.g., random seed selection			
Programming Style	e.g., $x+(y+z)$ vs. $(x+y)+z$			
Language				
Compiler & VM				
Library				
Platform				
Processor				
Micro-architecture	Inner state of			

Language	Library	System	Compiler	VariabilityPublic	EqualityCheck	NumberGenerations	Repeat	min	max	std	mean
Perl				seed None	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
Perl				seed None	MULT_INV	100	10	60.0	71.0	3.56230262611375	65.1
Perl				seed None	MULT_INV_P1	100	10	51.0	63.0	3.330165161669345	55.9
Perl				seed 42	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
Perl				seed 42	MULT_INV	100	10	62.0	62.0	0.0	62.0
Perl				seed 42	MULT_INV_P1	100	10	47.0	47.0	0.0	47.0
Go				seed None	associativity	100	10	71.0	82.0	3.3466401061363023	76.0
Go				seed None	multi-inverse	100	10	58.0	78.0	6.0	66.0
Go				seed None	multi-inverse-pi	100	10	42.0	64.0	5.88575587824865	53.4
Go				seed 42	associativity	100	10	81.0	81.0	0.0	81.0
Go				seed 42	multi-inverse	100	10	70.0	70.0	0.0	70.0
Go				seed 42	multi-inverse-pi	100	10	56.0	56.0	0.0	56.0
R				seed None	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
R				seed None	MULT_INV	100	10	62.0	72.0	2.764054992217051	66.6
R				seed None	MULT_INV_P1	100	10	47.0	57.0	2.808914381037628	51.1
R				seed 42	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
R				seed 42	MULT_INV	100	10	67.0	67.0	0.0	67.0
R				seed 42	MULT_INV_P1	100	10	53.0	53.0	0.0	53.0
julia				seed None strict-equality	ASSOCIATIVITY	100	10	74.0	90.0	4.60972228646435	82.5
julia				seed None strict-equality	MULT_INV	100	10	60.0	79.0	6.16765757804371	68.6
julia				seed None strict-equality	MULT_INV_P1	100	10	49.0	59.0	2.8301943396169813	54.3
julia				seed 42 strict-equality	ASSOCIATIVITY	100	10	89.0	89.0	0.0	89.0
julia				seed 42 strict-equality	MULT_INV	100	10	73.0	73.0	0.0	73.0
julia				seed 42 strict-equality	MULT_INV_P1	100	10	55.0	55.0	0.0	55.0
julia				seed None approximate equality of julia lang	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
julia				seed None approximate equality of julia lang	MULT_INV	100	10	100.0	100.0	0.0	100.0

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

AGENDA

Frictionless Reproducibility and (Deep) Software (Variability)

Problem (cont'd): Variability and Frictions

Solution: Variability and Exploration

Discussions

deep software variability



hardware variability

15,000+ options

thousands of compiler flags and compile-time options

dozens of preferences

100+ command-line parameters

1000+ feature toggles

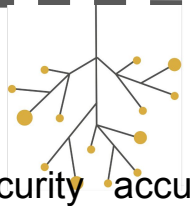
38

execution time

energy consumption

security

accuracy



Non-functional properties

deep software variability

hardware variability

15,000+ options

thousands of compiler flags
and compile-time options

dozens of preferences

input data

100+ command-line parameters

1000+ feature toggles

System under
Study
(reproducible
and
replicable)

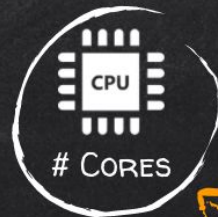
Variability
Output

(scientific result;
most of the time
quantitative
information)



DEEP VARIABILITY

HARDWARE



OPERATING SYSTEM



SOFTWARE

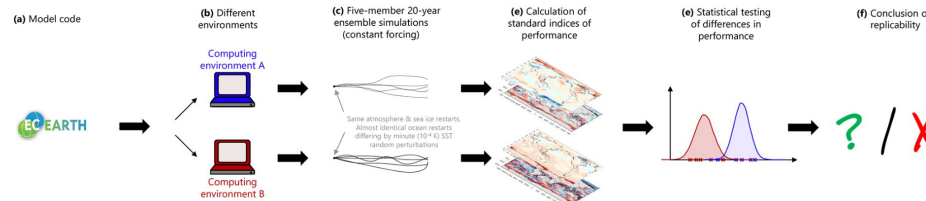


INPUT DATA



Replicability of the EC-Earth3 Earth system model under a change in computing environment

François Massonnet^{1,2}, Martin Ménégoz^{2,3}, Mario Acosta², Xavier Yepes-Arbós², Eleftheria Exarchou², and Francisco J. Doblas-Reyes^{2,4}



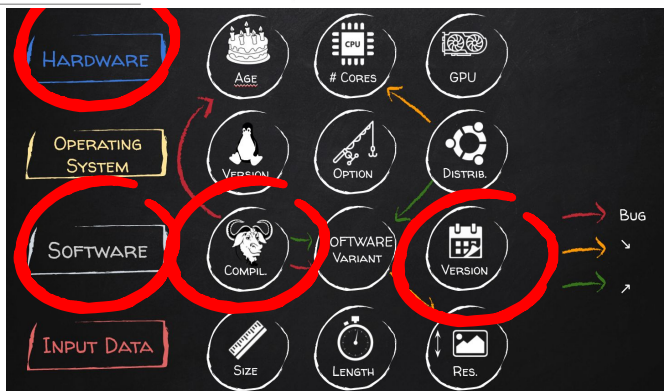
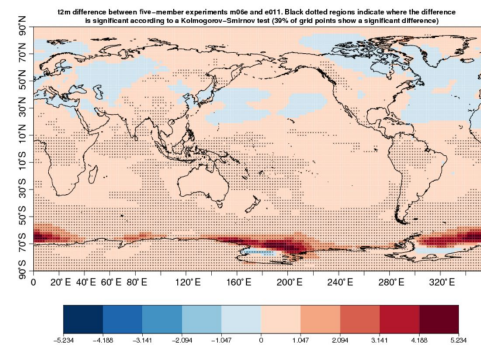
Can a coupled **ESM** simulation be restarted from a different machine without causing climate-changing modifications in the results? Using two versions of EC-Earth: one “**non-replicable**” case (see below) and one replicable case.

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

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

Table 2. The four experiments considered in this study.

Experiment ID	e011	m06e	a0gi	a0go
Computing environment	ECMWF-CCA	MareNostrum3	ECMWF-CCA	MareNostrum3
EC-Earth version	3.1	3.1	3.2	3.2
Processors (IFS+NEMO+OASIS)	598 (480 + 96 + 22)	512 (384 + 96 + 22)	432 (288 + 144)	416 (288 + 128)
F flags	-O2 -g -traceback -vec-report0 -r8 -vec-report0 -r8	-O2 -g -traceback -vec-report0 -r8 -vec-report0 -r8	-O2 -g -traceback -r8 -fp-model strict -fp-model strict -xHost	-O2 -fp-model precise -xHost -g -traceback -g -traceback -r8
C flags	-O2 -g -traceback	-O2 -g -traceback	-traceback -fp model strict -xHost	-O2 -fp-model precise -xHost -g -traceback
LD flags	-O2 -g -traceback	-O2 -g -traceback	-traceback -fp-model strict -xHost	-O2 -fp-model precise -xHost -g -traceback
Output size	141.8 GB	141.6 GB	101.3 GB	101.3 GB



Explanatory variable	Meaning
<i>entcoef</i>	Entrainment coefficient
<i>ct</i>	Accretion constant
<i>rhcrit</i>	Critical relative humidity
<i>vf1</i>	Ice fall speed through clouds
<i>eacf</i>	Empirically adjusted cloud fraction
<i>cw</i>	Threshold for precipitation
<i>dtice</i>	Temperature range of ice albedo variation
<i>ice</i>	Nonspherical ice
<i>midware</i>	Client middleware
<i>ice_size</i>	Ice particle size
<i>alphan</i>	Albedo at melting point of ice
<i>processor_name</i>	CPU classification
<i>clock_classic</i>	Processor clock speed recorded under classic middleware
<i>ram_size</i>	Hardware RAM
<i>clock_boinc_i</i>	Integer processor clock speed recorded under BOINC middleware
<i>clock_boinc_f</i>	Floating point processor clock speed recorded under BOINC middleware
<i>os_name</i>	Operating system
<i>dtheta</i>	Perturbations to initial conditions on a given level

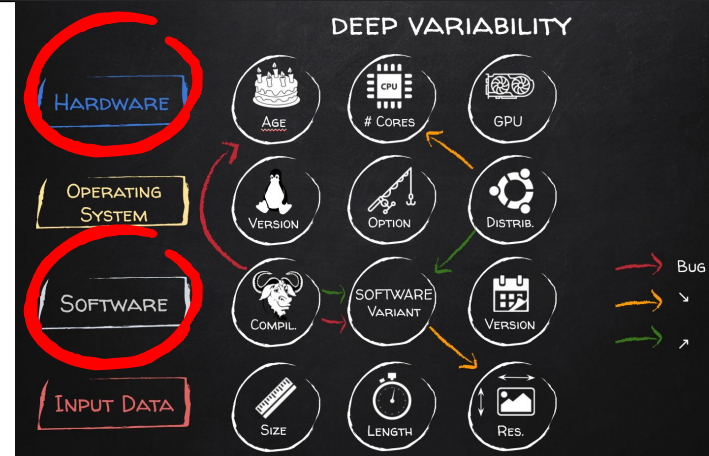
Association of parameter, software, and hardware variation with large-scale behavior across 57,000 climate models

Christopher G. Knight, Sylvia H. E. Knight, Neil Massey, Tolu Aina, Carl Christensen, Dave J. F...

[+ See all authors and affiliations](#)

PNAS July 24, 2007 104 (30) 12259-12264; <https://doi.org/10.1073/pnas.0608144104>

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

Joelle Pineau “Building Reproducible, Reusable, and Robust Machine Learning Software” ICSE’19 keynote “[...] results can be brittle to even minor perturbations in the domain or experimental procedure”

Deep Reinforcement Learning that Matters

Peter Henderson^{1*}, Riashat Islam^{1,2*}, 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?

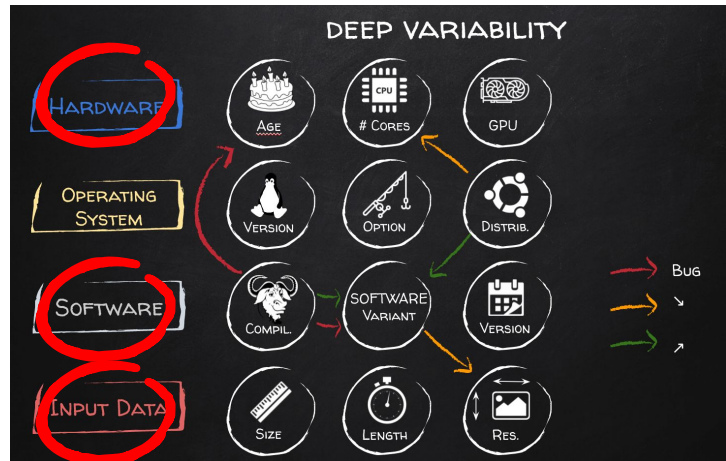
torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision

David Picard
LIGM, École des Ponts, 77455 Marnes la vallée, France

DAVID.PICARD@ENPC.FR

Abstract

In this paper I investigate the effect of random seed selection on the accuracy when using popular deep learning architectures for computer vision. I scan a large amount of seeds (up to 10^4) on CIFAR 10 and I also scan fewer seeds on Imagenet using pre-trained models to investigate large scale datasets. The conclusions are that even if the variance is not very large, it is surprisingly easy to find an outlier that performs much better or much worse than the average.



Reproducible and replicable CFD: it's harder than you think

Olivier Mesnard, Lorena A. Barba

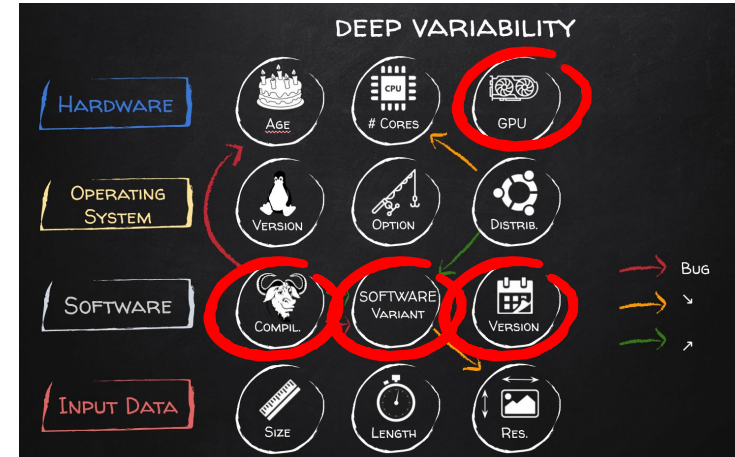
Mechanical and Aerospace Engineering, George Washington University, Washington DC 20052

“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



Variability in the analysis of a single neuroimaging dataset by many teams

[Rotem Botvinik-Nezer](#), [Felix Holzmeister](#), ... [Tom Schonberg](#)  [+ Show authors](#)

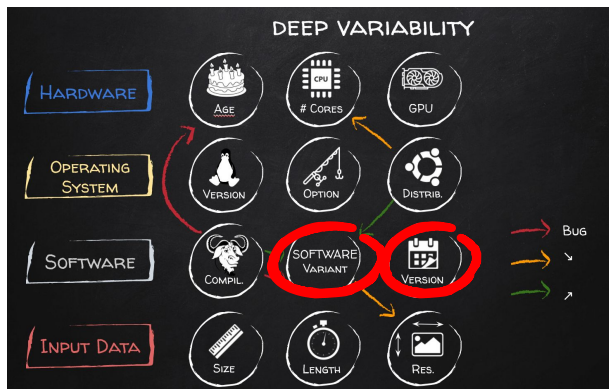
[Nature](#) **582**, 84–88 (2020) | [Cite this article](#)

42k Accesses | **203** Citations | **2056** Altmetric | [Metrics](#)

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.”



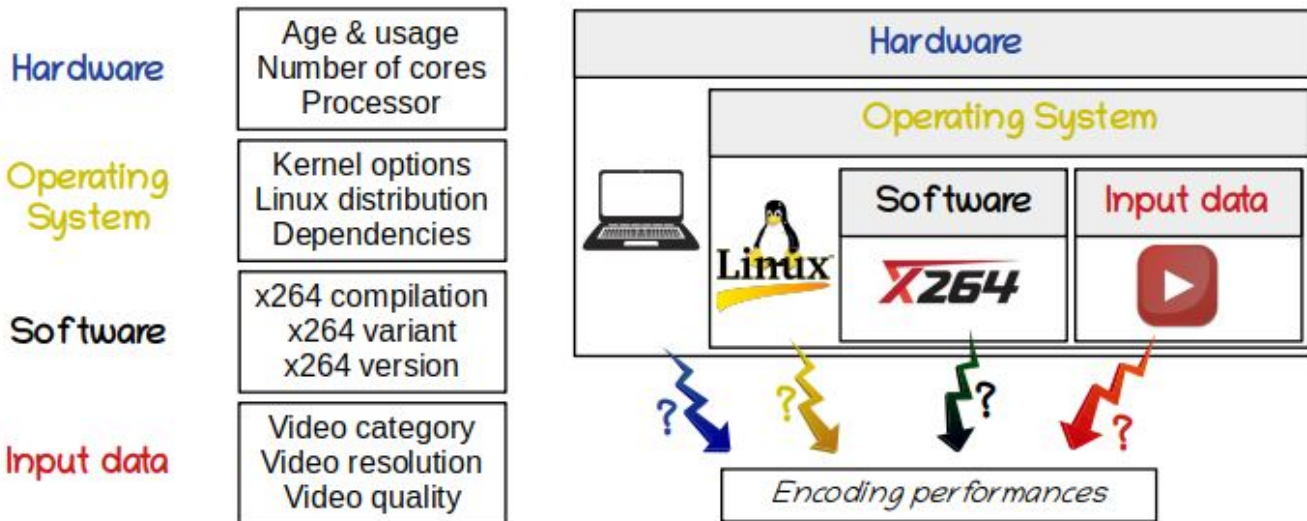
In this paper, we empirically studied 11,363 ML pipelines from diverse competitions on KAGGLE to explore the impacts of different ML library version combinations on their performances. Our study reveals the pervasiveness and severity of PLC issues in ML pipelines. Our findings can motivate the establishment of a symbiotic ecosystem where researchers, tool builders, and library vendors work together to assist developers in combating PLC issues.

{Keras, Tensorflow} Versions	Score (AUC)	Ranking	Time (ms)	Memory (MB)
{2.7.1, 2.7.0}	Crash	Crash	Crash	Crash
{2.4.3, 2.4.1}	0.768	522	1895.656	1244.446
{2.4.3, 2.3.1}	0.737	521	1882.099	1241.001
{2.4.3, 2.2.0}	0.559	523	1926.980	1248.518
{2.3.1, 2.4.1}	Crash	Crash	Crash	Crash
{2.3.1, 2.3.1}	Crash	Crash	Crash	Crash
{2.3.1, 2.2.0}	0.997	1	1877.330	1199.130
{2.3.1, 2.1.0}	0.997	1	1888.612	1202.602
{2.3.1, 2.0.0}	Crash	Crash	Crash	Crash
{2.3.1, 1.15.2}	0.997	1	1989.861	1194.425
{2.3.1, 1.14.0}	0.997	1	1853.423	1196.269
{2.3.1, 1.13.1}	0.997	1	1901.693	1183.123

Deep software variability: Are layers/features orthogonal or are there interactions?

Variability layers

x264 environment



Luc Lesoil, Mathieu Acher, Arnaud Blouin, Jean-Marc Jézéquel:
Deep Software Variability: Towards Handling Cross-Layer Configuration.

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
```

480

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-bufsize 2000 --bitrate 1000 -o <output> <input>
```

Presets:

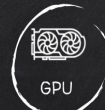
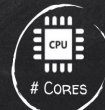
```
--profile <string> Force the limits of an H.264 profile
                    Overrides all settings.
                    - baseline, main, high, high10, high422
--preset <string> Use a preset to select encoding settings [medium]
                    Overrides by user settings.
                    - ultrafast,superfast,veryfast,faster,fast
                    - medium,slow,slower,veryslow,placebo
--tune <string> Tune the settings for a particular type of source
                or situation
                    Overrides by user settings.
                    Multiple tunings are separated by commas.
                    Only one psy tuning can be used at a time.
```

```
-I, --keyint <integer or "infinite"> Maximum GOP size [250]
--tff Enable interlaced mode (top field first)
--bff Enable interlaced mode (bottom field first)
--pulldown <string> Use soft pulldown
                    - none, 22, 3
```

Ratecontrol:

```
-B, --bitrate <integer> Set bitrate (kbit/s)
--crf <float> Quality-based VBR
--vbv-maxrate <integer> Max local bitrate
--vbv-bufsize <integer> Set size of the VBV buffer
-p, --pass <integer> Enable multipass encoding
                    - 1: First pass
                    - 2: Last pass
```

HARDWARE



OPERATING SYSTEM



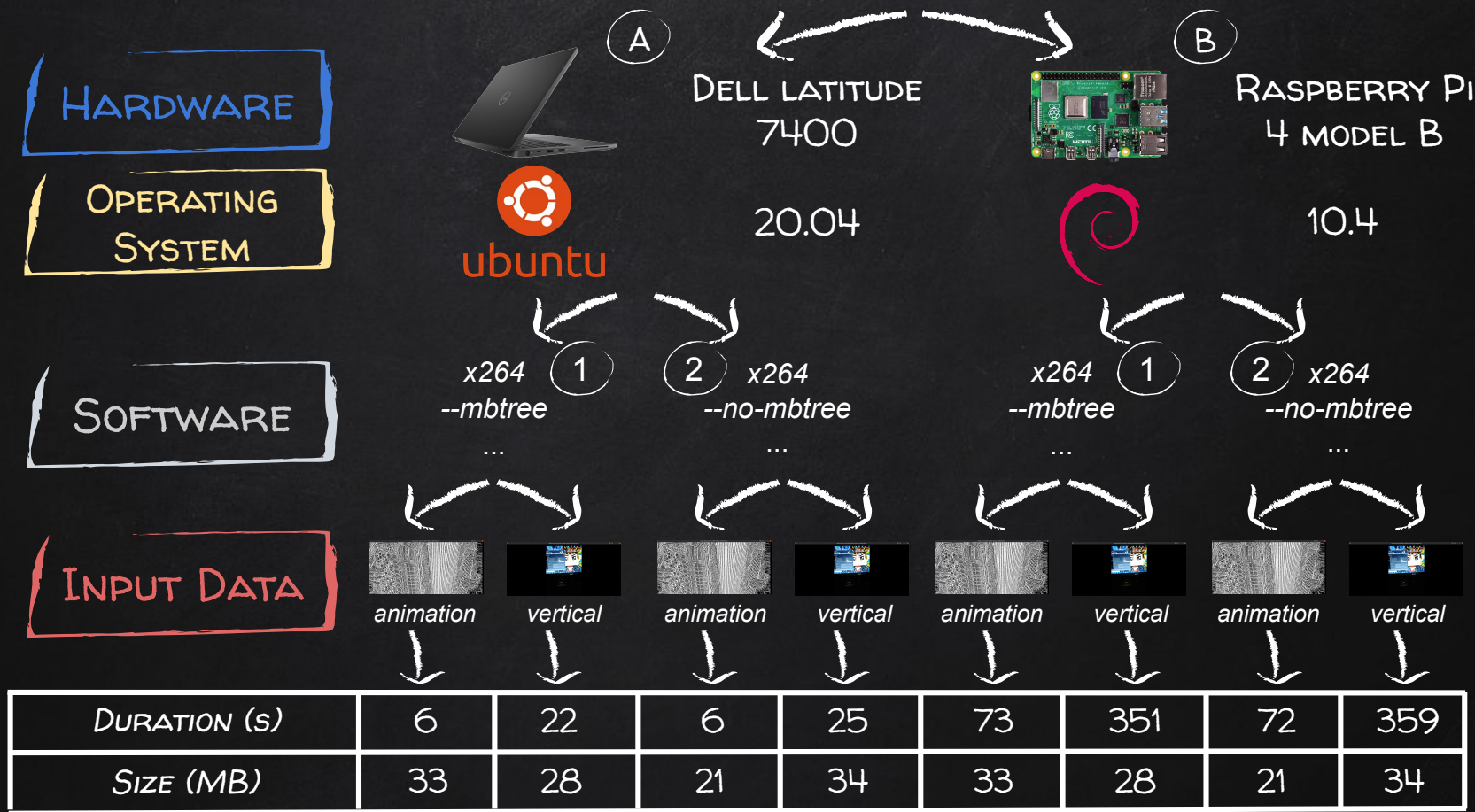
SOFTWARE



INPUT DATA



REAL WORLD EXAMPLE (x264)

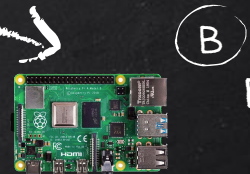


REAL WORLD EXAMPLE (x264)

HARDWARE



(A) DELL LATITUDE 7400



(B) RASPBERRY PI 4 MODEL B

OPERATING SYSTEM



20.04



10.4

SOFTWARE

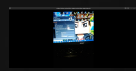
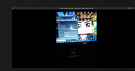
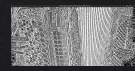
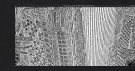
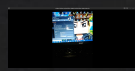
x264 (1) --mbtree

(2) x264 --no-mbtree

x264 (1) --mbtree

(2) x264 --no-mbtree

INPUT DATA



animation

vertical

animation

vertical

animation

vertical

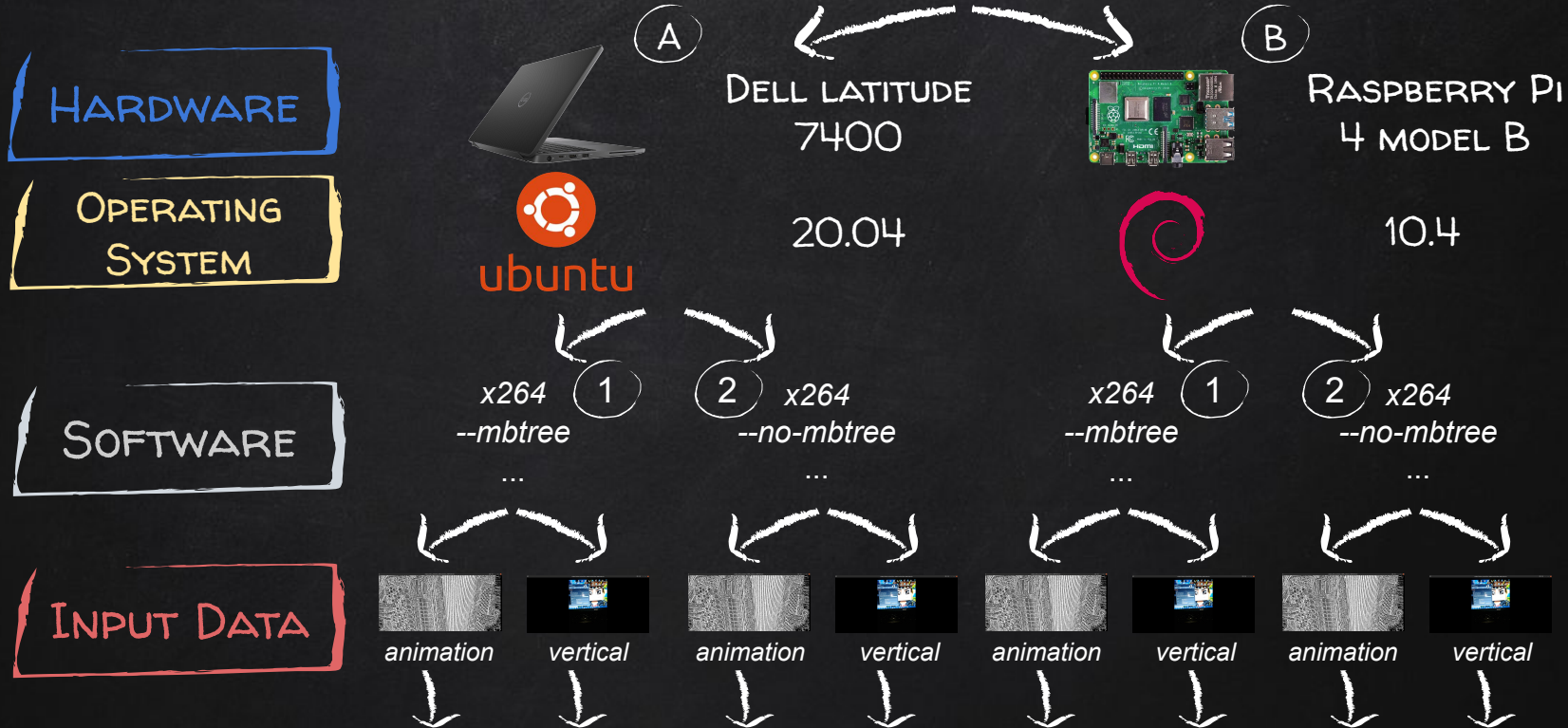
animation

vertical

DURATION (s)	6	22	6	25	73	351	72	359
SIZE (MB)	33	28	21	34	33	28	21	34

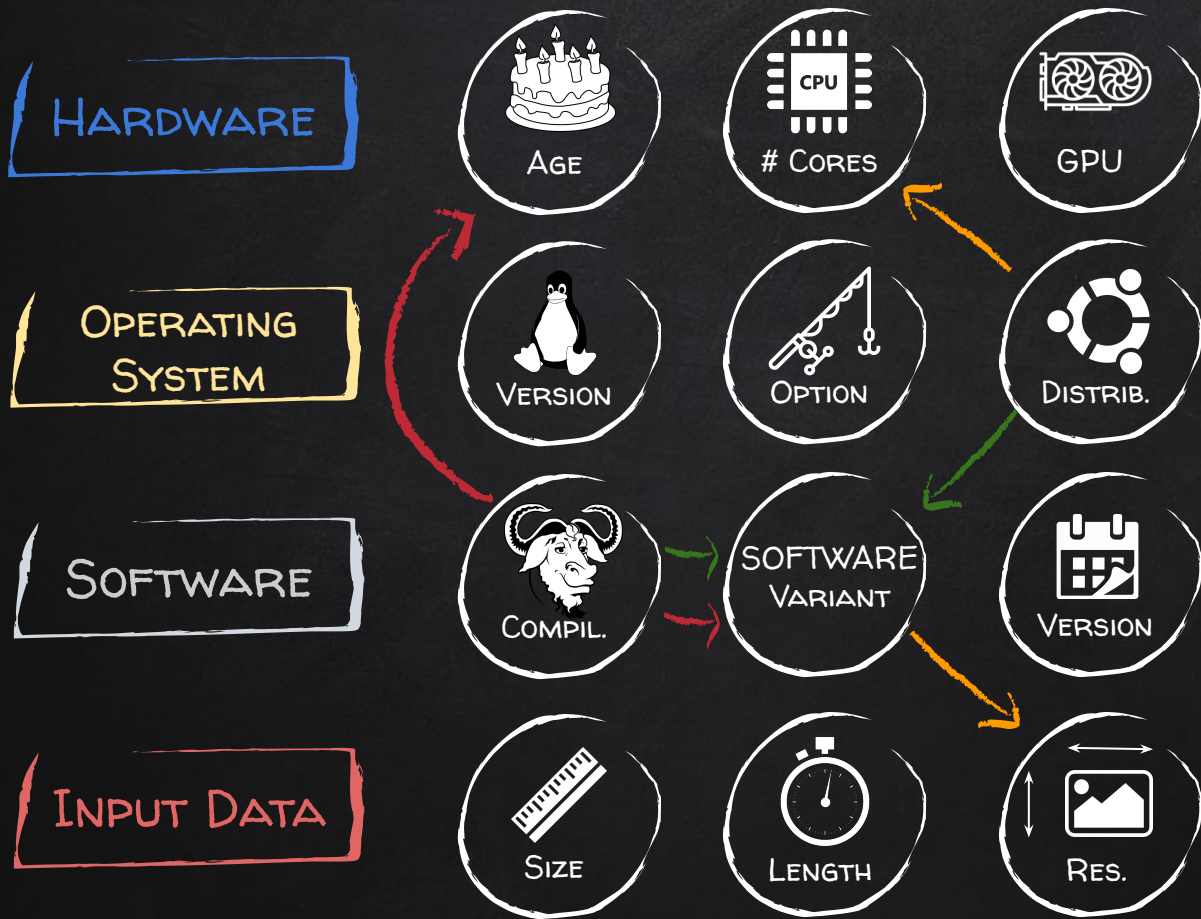


REAL WORLD EXAMPLE (x264)



DURATION (s)	6	22	6	25	73	351	72	359	≈*16
SIZE (MB)	33	28	21	34	33	28	21	34	≈*12

DEEP VARIABILITY



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**



L. Lesoil, M. Acher, A. Blouin and J.-M. Jézéquel, “Deep Software Variability: Towards Handling Cross-Layer Configuration” in VaMoS 2021

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?

two performance models f_1 and f_2

$$f_1 = \beta \times f_2 + \alpha$$



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

configurationID	cabac	ref	mixed_ref	me	subme	me_range	trellis	elapsedtime	fps	rank_elapsedtime
1	0	1	0	dia	0	16	0	0:02:14	375.22	1
138	0	5	0	tesa	10	24	2	0:04:54	155.35	7
15	0	1	0	dia	0	16	0	0:02:22	384.22	3
16	0	1	0	dia	0	16	0	0:02:24	375.4	4
17	0	1	0	hex	0	16	0	0:02:19	385.92	2
21	0	1	0	dia	0	16	0	0:02:84	260.65	6
22	0	1	0	dia	0	16	0	0:02:61	303.2	5

configurations' measurements over input_42

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

Inputs =



configurations' measurements over input_1

configurationID	cabac	ref	mixed_ref	me	subme	me_range	trellis	elapsedtime	fps	rank_elapsedtime
1	0	1	0	dia	0	16	0	0:02:14	375.22	1
138	0	5	0	tesa	10	24	2	0:04:54	155.35	7
15	0	1	0	dia	0	16	0	0:02:22	384.22	3
16	0	1	0	dia	0	16	0	0:02:24	375.4	4
17	0	1	0	hex	0	16	0	0:02:19	385.92	2
21	0	1	0	dia	0	16	0	0:02:84	260.65	6
22	0	1	0	dia	0	16	0	0:02:61	303.2	5

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

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

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?

two performance models f_1 and f_2

$$f_1 = \beta \times f_2 + \alpha$$



configurationID	cabac	ref	mixed_ref	me	subme	me_range	trellis	elapstime	fps	rank_elapstime
1	0	1	0	dia	0	16	0	0:02:14	375.22	1
138	1	0	0	dia	0	16	0	04:37	375.22	3
15	1	0	5	tesa	10	24	2	07:56	155.35	7
16	0	1	0	dia	0	16	0	07:23	384.22	6
17	0	1	0	dia	0	16	0	04:33	375.4	2
21	0	1	0	hex	0	16	0	06:00	385.92	5
22	0	1	0	dia	0	16	0	05:48	260.65	4
22	0	1	0	dia	0	16	0	02:19	303.2	1

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?



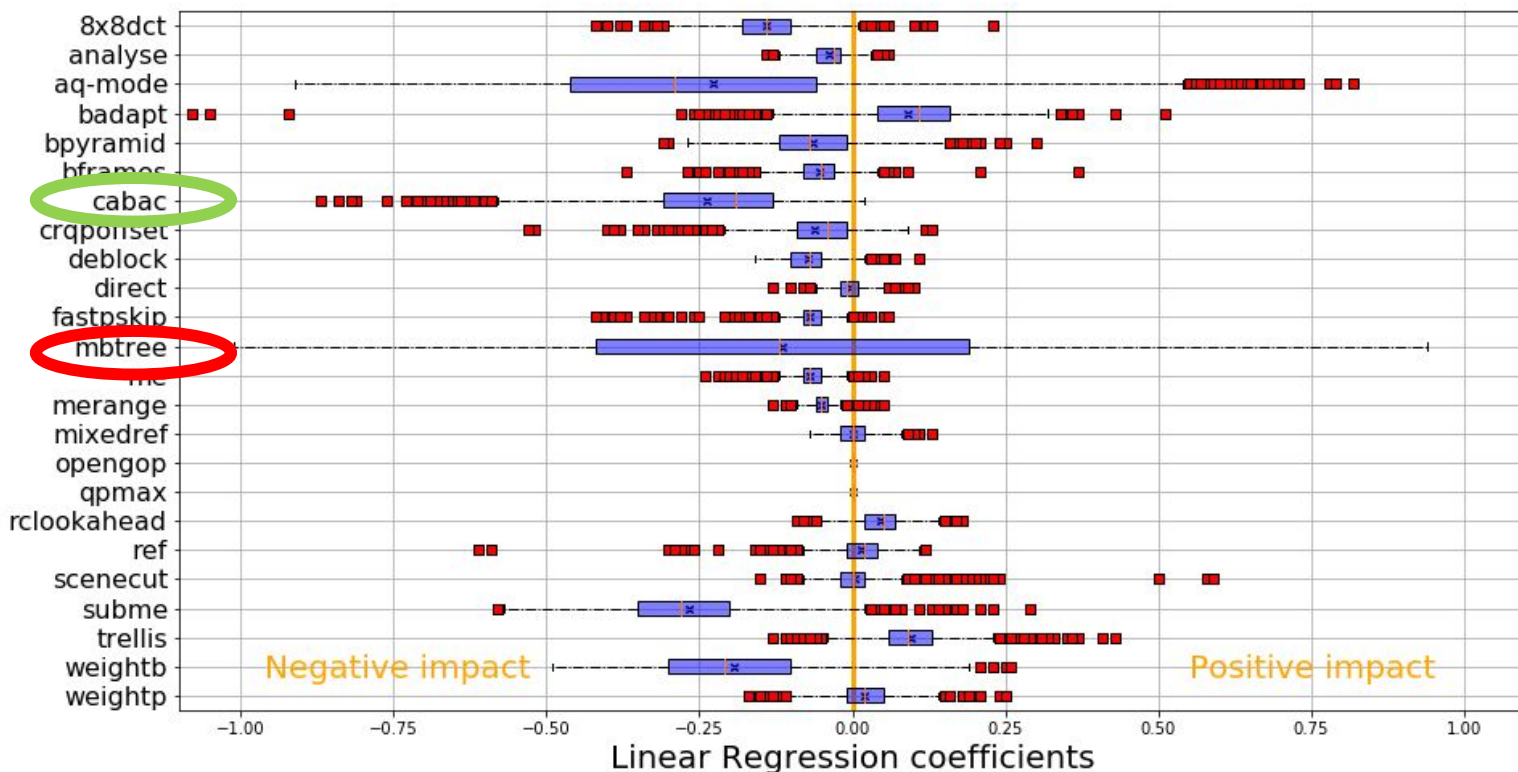
1397 videos x 201 software configurations

- 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

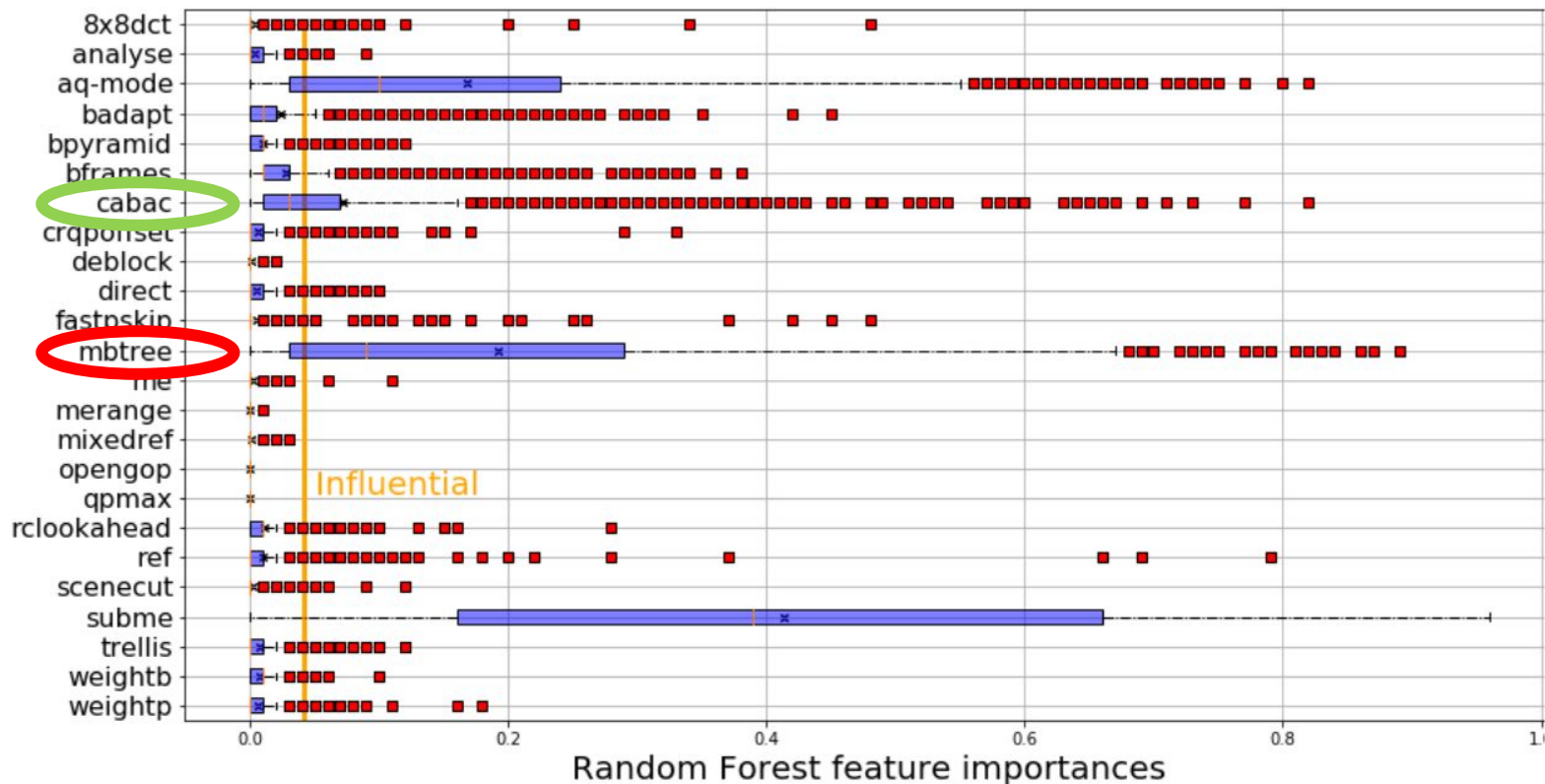
two performance models f_1 and f_2

$$f_1 = \beta \times f_2 + \alpha \quad ?$$

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!

System	Domain	Commit	Configs #C	Inputs I	#I
gcc	Compilation	ccb4e07	80	.c programs	30
ImageMagick	Image processing	5ee49d6	100	images	1000
lingeling	SAT solver	7d5db72	100	SAT formulae	351
nodeJS	JS runtime env.	78343bb	50	.js scripts	1939
poppler	PDF rendering	42dde68	16	.pdf files	1480
SQLite	DBMS	53fa025	50	databases	150
x264	Video encoding	e9a5903	201	videos	1397
xz	Data compression	e7da44d	30	system files	48

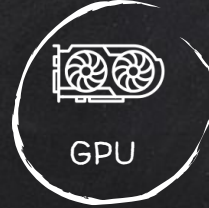
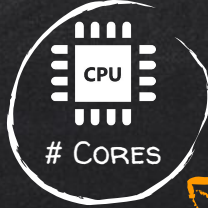
System	#M	Performance(s) P	Docker	Dataset
gcc	2400	size, ctime, exec	Link	Link
ImageMagick	100 000	size, time	Link	Link
lingeling	35 100	#confl.,#reduc.	Link	Link
nodeJS	96 950	#operations/s	Link	Link
poppler	23 680	size, time	Link	Link
SQLite	7500	15 query times q1-q15	Link	Link
x264	280 797	cpu, fps, kbs, size, time	Link	Link
xz	1440	size, time	Link	Link



L. Lesoil, M. Acher, A. Blouin and J.-M. Jézéquel “The Interaction between Inputs and Configurations fed to Software Systems: an Empirical Study”
<https://arxiv.org/abs/2112.07279>

DEEP VARIABILITY

HARDWARE



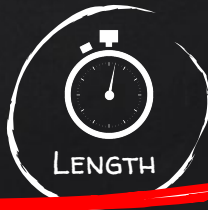
OPERATING SYSTEM



SOFTWARE



INPUT DATA



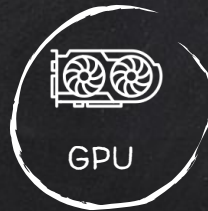
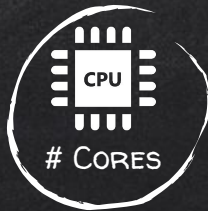
Sometimes, variability is consistent/stable and knowledge transfer is immediate.

But there are also interactions among variability layers and **variability knowledge may not generalize**

- BUG
- PERF. ↓
- PERF. ↗

DOES DEEP SOFTWARE VARIABILITY AFFECT PREVIOUS SCIENTIFIC, SOFTWARE-BASED STUDIES? (A GRAPHICAL TEMPLATE)

HARDWARE



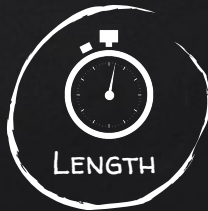
OPERATING SYSTEM



SOFTWARE



INPUT DATA



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)...

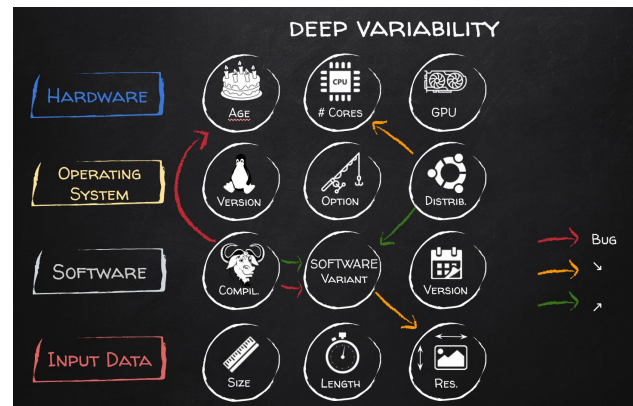
Or not desirable

- 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** envelope
- Enable **exploration** and **optimization**
- Innovation and new hypothesis, insights, **knowledge**



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

Our Vision

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

Embracing Deep Variability For Reproducibility & Replicability

Mathieu Acher, Benoit Combemale, Georges Aaron Randrianaina, 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 parties 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 and vision paper we delve into the application of software engineering techniques, specifically variability management, to systematically identify and explicit 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 hypotheses.

2 DEEP SOFTWARE VARIABILITY

Uncertainty in informatics comes from many different origins [16, 36], either ontological (*i.e.*, inherent unpredictability, *e.g.*, aleatory) 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 processor¹.

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.*, `-3%2` evaluates to `-1` in Java but to `1` in Python), or even inside the same programming language (for instance `x/0` is an undefined behavior in C).

Parameters	<i>e.g.</i> , random seed selection
Input Data	
Programming Style	<i>e.g.</i> , <code>x*(y+z)</code> vs. <code>(x+y)+z</code>

ACM REP 2024

⇒ 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”.

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

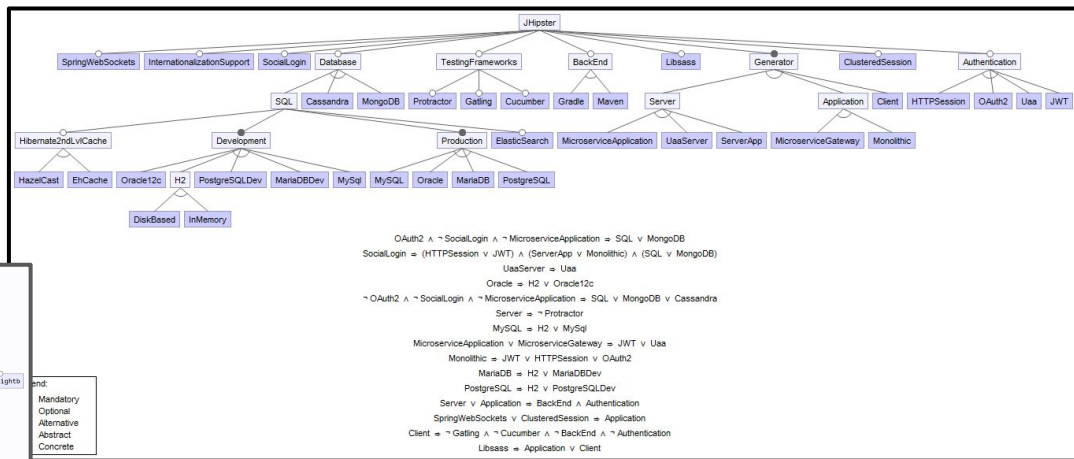
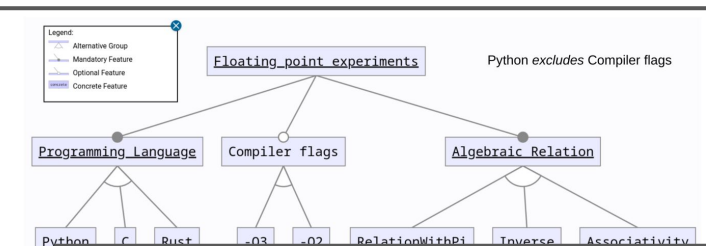
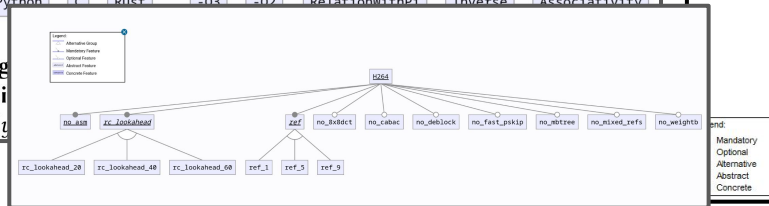


Fig
Wi
x/t



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)**

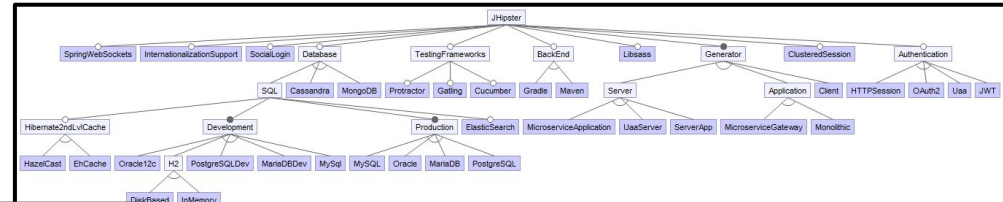
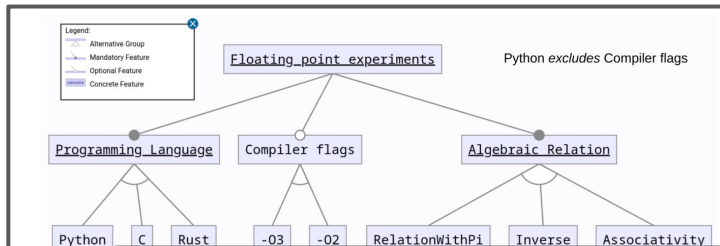
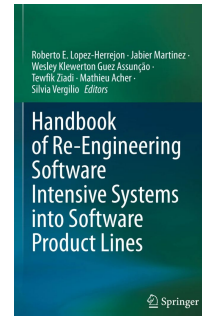
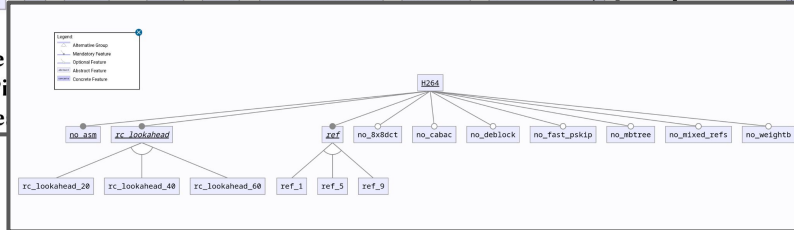


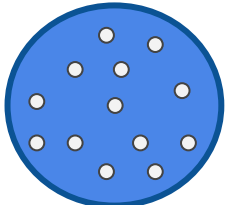
Figure WithPi x/y (re



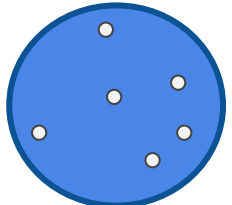
$OAuth2 \wedge \sim SocialLogin \wedge \sim MicroserviceApplication \Rightarrow SQL \vee MongoDB$
 $SocialLogin \Rightarrow (HTTPSession \vee JWT) \wedge (ServerApp \vee Monolithic) \wedge (SQL \vee MongoDB)$
 $UaaServer \Rightarrow Uaa$
 $Oracle \Rightarrow H2 \vee Oracle12c$
 $\sim OAuth2 \wedge \sim SocialLogin \wedge \sim MicroserviceApplication \Rightarrow SQL \vee MongoDB \vee Cassandra$
 $Server \Rightarrow \sim Protractor$
 $MySQL \Rightarrow H2 \vee MySQL$
 $MicroserviceApplication \vee MicroserviceGateway \Rightarrow JWT \vee Uaa$
 $Monolithic \Rightarrow JWT \vee HTTPSession \vee OAuth2$
 $MariaDB \Rightarrow H2 \vee MariaDBDev$
 $PostgreSQL \Rightarrow H2 \vee PostgreSQLDev$
 $Server \vee Application \Rightarrow Backend \wedge Authentication$
 $SpringWebSockets \vee ClusteredSession \Rightarrow Application$
 $Client \Rightarrow \sim Gating \wedge \sim Cucumber \wedge \sim Backend \wedge \sim Authentication$
 $Libsass \Rightarrow Application \vee Client$

Solution #2: sampling and learning

(regression, classification)



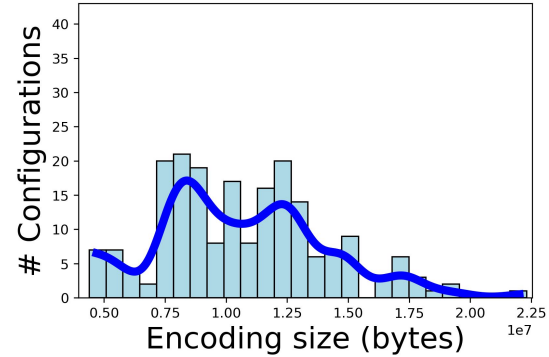
Whole Population of Configurations



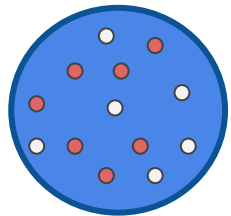
Training Sample



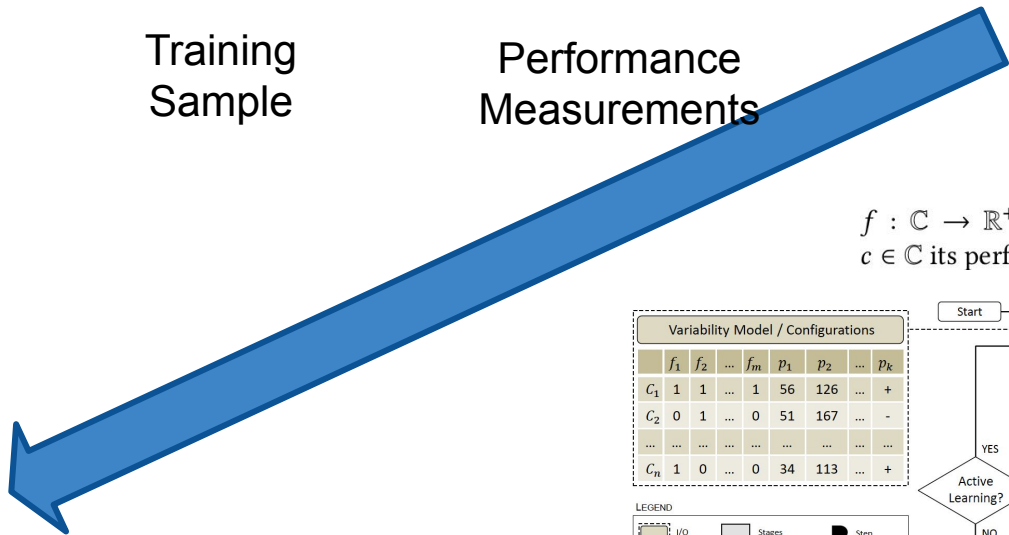
Performance Measurements



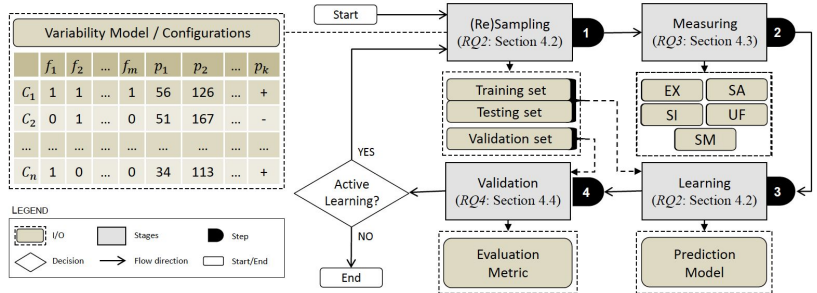
Prediction Model



Performance Prediction



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



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

```
x264 --me dia  
--ref 5
```

...

```
-o output_1.x264
```





hardware variability

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)

~~Variability~~
Output
(binary)

“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 Zacchiroli “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022

deep software variability



15,000+

compile-time options



System under
Study

~~Variability~~
Output
(binary)

“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 Zacchiroli “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?

Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems Randrianaina, Khelladi, Zendra, Acher MSR'2024

also at FOSDEM 2024 <https://fosdem.org/2024/schedule/event/fosdem-2024-2848-documenting-and-fixing-non-reproducible-builds-due-to-configuration-options/>

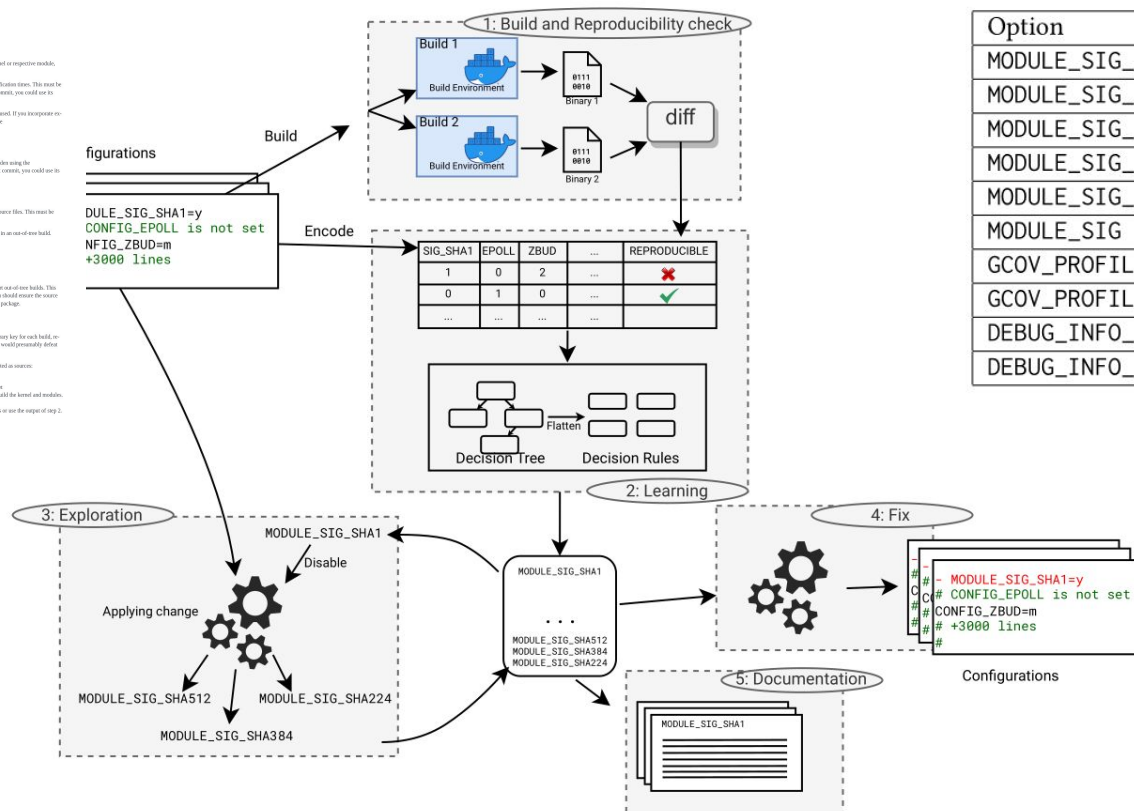
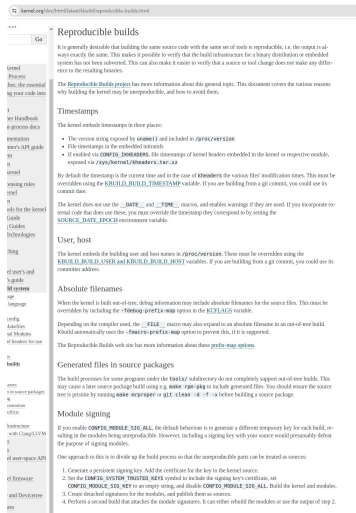


Table 2: Identified options and their category.

Option	Category
MODULE_SIG_SHA1	Module Signing
MODULE_SIG_SHA224	Module Signing
MODULE_SIG_SHA256	Module Signing
MODULE_SIG_SHA384	Module Signing
MODULE_SIG_SHA512	Module Signing
MODULE_SIG	Module Signing
GCOV_PROFILE_FTRACE	Profiling
GCOV_PROFILE_ALL	Profiling
DEBUG_INFO_SPLIT	Debug Info
DEBUG_INFO_REDUCED	Debug Info

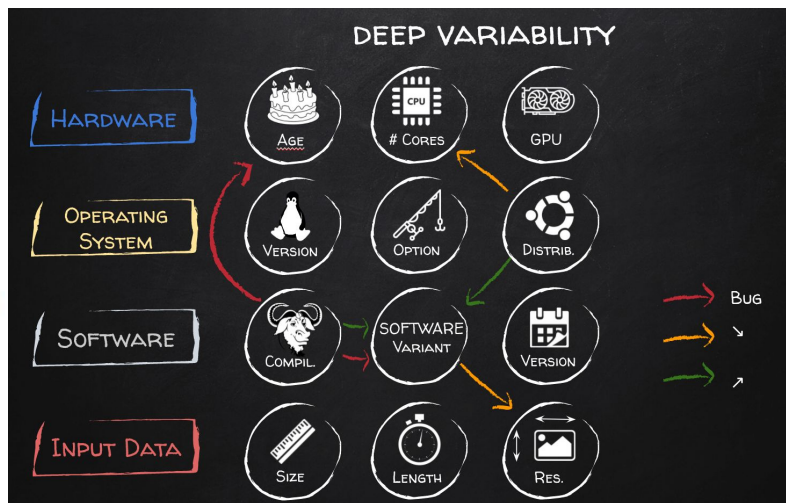


Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems

Randrianaina, Khelladi, Zendra, Acher MSR'2024

also at FOSDEM 2024 <https://fosdem.org/2024/schedule/event/fosdem-2024-2848-documenting-and-fixing-non-reproducible-builds-due-to-configuration-options/>

#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 Zacchiroli “Reproducible Builds: Increasing the Integrity of Software Supply Chains” IEEE Software 2022

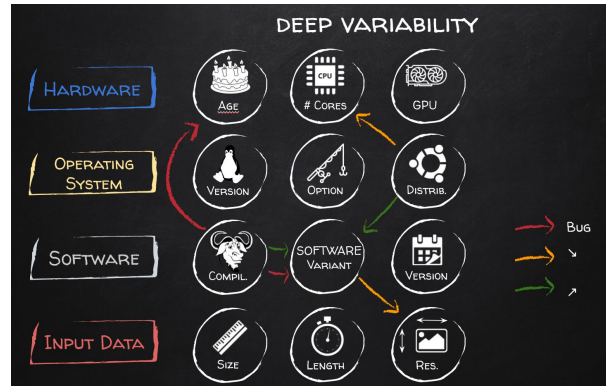
Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems

Randrianaina, Khelladi, Zendra, Acher MSR'2024

also at FOSDEM 2024 <https://fosdem.org/2024/schedule/event/fosdem-2024-2848-documenting-and-fixing-non-reproducible-builds-due-to-configuration-options/>

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

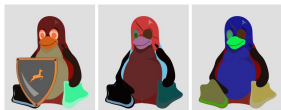
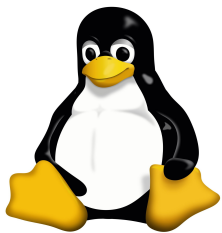
Busybox. To pinpoint the source of non-reproducible builds, the typical workflow is to slightly vary the build environment. Changing the build path between two builds of the same configuration for Busybox impacts 49.75% of the configurations, causing their build to be non-reproducible (presented in Figure 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. 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.

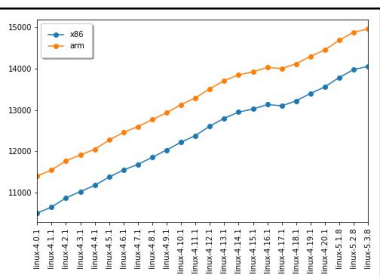


“The build process of a software product is reproducible if, after designating a specific version **and a specific variant** of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed.” Lamb and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



- 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...

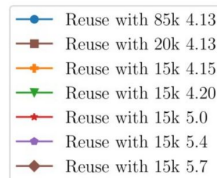
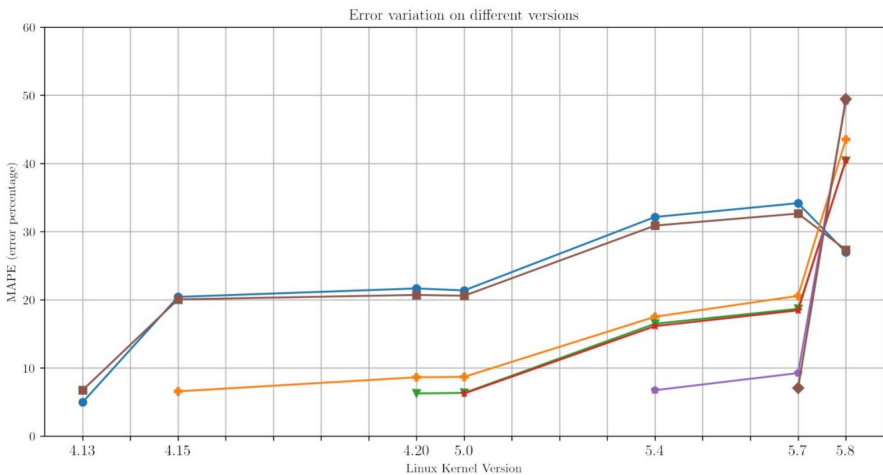




Version	Release Date	LOC	Files	Examples	Seconds/config	Options	Features	Deleted features	New features	Δ Commits	Files changes
4.13	2017/09/03	16,616,534	60,530	92,562	not available	12,776	9,468	-	-	-	-
4.15	2018/01/28	17,073,368	62,249	39,391	not available	12,998	9,425	342	299	31,052	934,628
4.20	2018/12/23	17,526,171	62,423	23,489	225	13,533	10,189	468	1,189	104,691	1,972,020
5.0	2019/03/03	17,679,372	63,076	19,952	247	13,673	10,293	494	1,319	118,778	2,170,935
5.4	2019/10/24	19,358,903	67,915	25,847	285	14,159	10,813	663	2,008	181,308	3,827,025
5.7	2020/05/31	19,358,903	67,915	20,159	258	14,586	11,338	715	2,585	225,804	4,393,117
5.8	2020/08/02	19,729,197	69,303	21,923	289	14,817	11,530	730	2,792	242,381	4,681,313

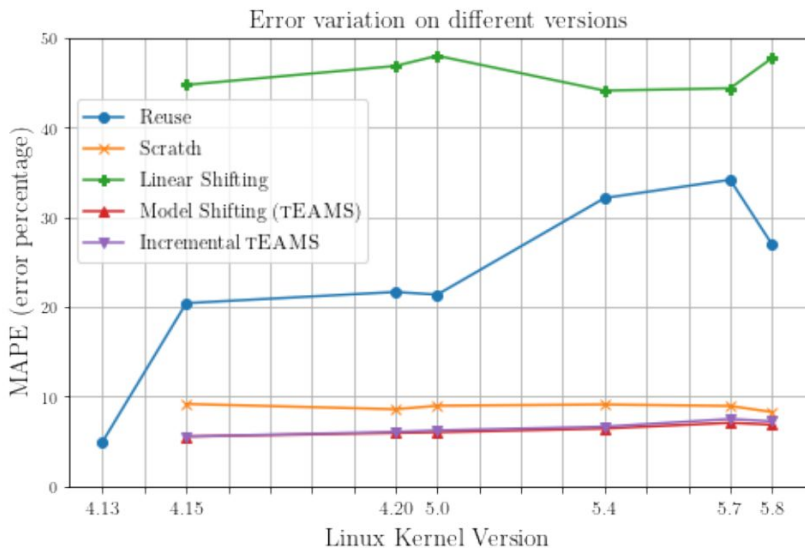
Table I: Dataset properties for each version. The number of deleted/new features, delta commits, files changes are w.r.t. 4.13.

4.13 version (sep 2017): 6%. What about **evolution?** Can we reuse the 4.13 Linux prediction model? No, accuracy quickly decreases: **4.15 (5 months after): 20%; 5.7 (3 years after): 35%**



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:
 Learning input-aware performance models of configurable systems: An empirical evaluation. J. Syst. Softw. 208: 111883 (2024)

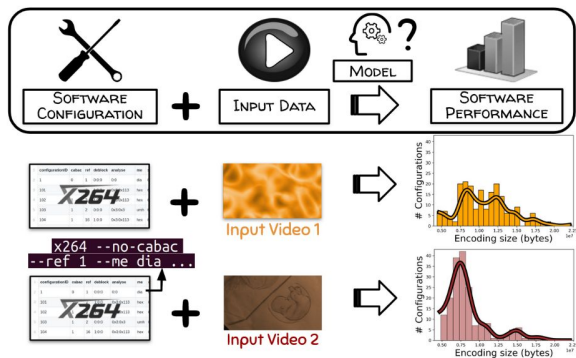
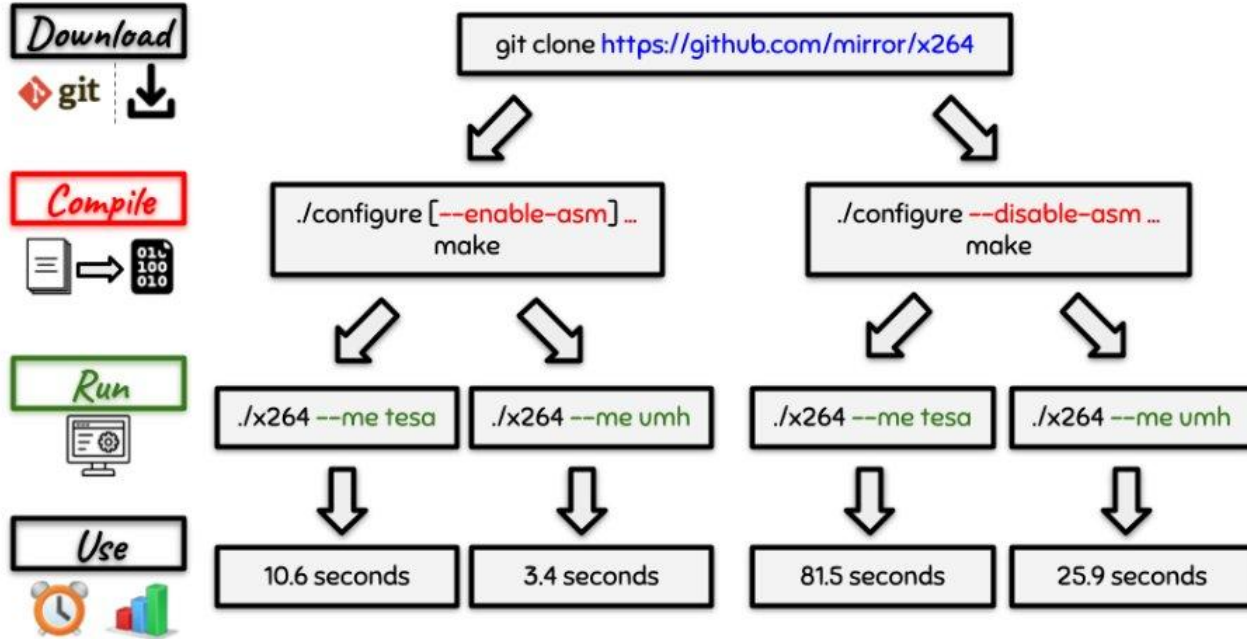


Figure 1: The performance prediction problem: how to predict software performance considering both configurations and inputs?

Approach	Description of the approach	Offline cost (Offshore Organization)	Online measurement cost (User)	Input properties
Supervised online learning	Train performance model on demand, from scratch, each time a new input is fed to the configurable system	None	High	No
Offline learning	Use a pre-trained model over measurements of multiple configurations and inputs. Input properties are used to make the prediction (in an online setting).	High	None	Yes
Transfer learning	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).	Medium	Medium	Yes

Is there an interplay between **compile-time** and **runtime** options?

L. Lesoil, M. Acher, X. Těrnava, A. Blouin and J.-M. Jézéquel "The Interplay of Compile-time and Run-time Options for Performance Prediction" in SPLC '21

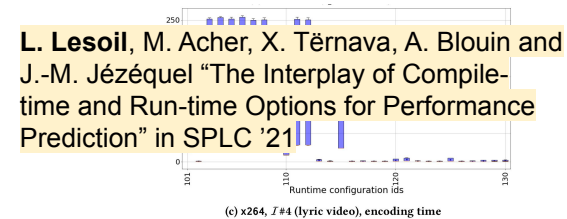


This paper investigates how compile-time options can affect software performances and how compile-time options interact with run-time options.

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 aw
3. Measuring at lower cost: do not use a default compile-time configuration, use it will generalize!

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



Key results (Poppler)

First good news: Worth it

Second good news: correlation is weak. It does not change the rankings

It has three practical implications:



XZ

performances!

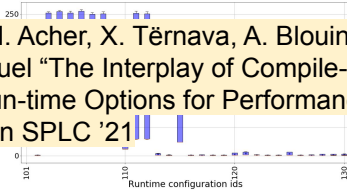
interplay between compile-time and runtime options and even input!

transfer learning of... Users can... the configuration is...

...st compile-time configuration among all the possible ones allows configuration at run time. We can remove away... a default compile-time configuration, use



L. Lesoil, M. Acher, X. Těrnava, A. Blouin and J.-M. Jézéquel "The Interplay of Compile-time and Run-time Options for Performance Prediction" in SPLC '21



(c) x264, T #4 (lyric video), encoding time

...all the input videos, the worst scale of the distribution, they do not interact with the runtime options).



What
is your move?



What is your prompt?

[Event "FIDE World
Championship Match 2024"]
[Site "Los Angeles, USA"]
[Date "2024.12.01"]
[Round "5"]
[White "Kramnik, Vladimir"]
[Black "Nepomniachtchi,
Ian"]
[Result "1-0"]
[WhiteElo "2900"]
[BlackElo "2900"]
[TimeControl
"40/7200:20/3600:900+30"]
[UTCDate "2024.11.27"]
[UTCTime "09:01:25"]
[Variant "Standard"]



[Event "FIDE World
Championship Match 2024"]
[Site "Los Angeles, USA"]
[Date "2024.12.01"]
[Round "5"]
[White "Louapre, David"]
[Black "Giraud, Thibaut"]
[Result "0-1"]
[WhiteElo "1400"]
[BlackElo "1400"]
[TimeControl
"40/7200:20/3600:900+30"]
[UTCDate "2024.11.27"]
[UTCTime "09:01:25"]
[Variant "Standard"]





MrPhi
@MonsieurPhi

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).

[Translate post](#)

Playground

Complete

Playground

Complete

[Event "Chess Tournament"]

[Site "?"]

[Date "2024.0115"]

[Round "1"]

[White "Louapre, David"]

[Black "Carlsen, Magnus"]

[Result "1-0"]

[WhiteElo "1800"]

[BlackElo "1800"]

1.e4 e5 2.Bc4 Nc6 3.Qh5 g6

[Event "Chess Tournament"]

[Site "?"]

[Date "2024.0115"]

[Round "1"]

[White "Louapre, David"]

[Black "Carlsen, Magnus"]

[Result "0-1"]

[WhiteElo "1800"]

[BlackElo "1800"]

1.e4 e5 2.Bc4 Nc6 3.Qh5 Nf6

11:55 AM · Apr 19, 2024 · 447 Views

1



7



Post your reply

Reply



MrPhi @MonsieurPhi · Apr 19

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.

2



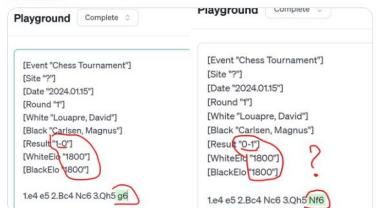
9

441



MrPhi @MonsieurPhi

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).



11:55 AM · Apr 19, 2024 · 447 Views

1 7

Post your reply Reply

MrPhi @MonsieurPhi · Apr 19

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.

2 441

```
# 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"""[Event "FIDE World Championship Match 2024"]
[Site "Los Angeles, USA"]
[Date "2024.12.01"]
[Round "5"]
[White "{white_name}"]
[Black "{black_name}"]
[Result "{result}"]
[WhiteElo "{white_elo}"]
[BlackElo "{black_elo}"]
[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
)
```





MrPhi
@MonsieurPhi

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).

[Translate post](#)

Playground Complete

[Event "Chess Tournament"]
[Site "?"]
[Date "2024.01.15"]
[Round "1"]
[White "Louapre, David"]
[Black "Carlsen, Magnus"]
[Result "1-0"]
[WhiteElo "1800"]
[BlackElo "1800"]

1.e4 e5 2.Bc4 Nc6 3.Qh5 g6

[Event "Chess Tournament"]
[Site "?"]
[Date "2024.01.15"]
[Round "1"]
[White "Louapre, David"]
[Black "Carlsen, Magnus"]
[Result "0-1"]
[WhiteElo "1800"]
[BlackElo "1800"]

1.e4 e5 2.Bc4 Nc6 3.Qh5 Nf6

11:55 AM · Apr 19, 2024 · 447 Views

1 7



Post your reply

Reply

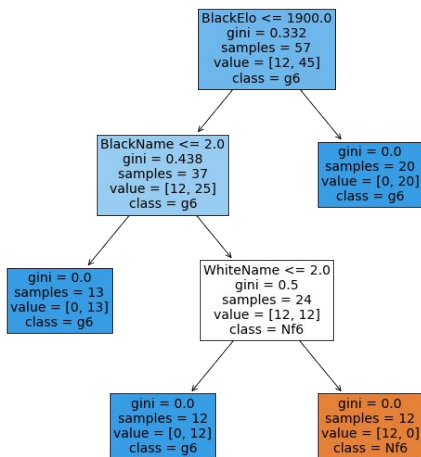


MrPhi @MonsieurPhi · Apr 19

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.

2 441

Result	WhiteName	BlackName	WhiteElo	BlackElo	IncludeWhiteTitle	IncludeBlackTitle	Move
0-1	XXX	Louapre, David	1800	1800	False	False	Nf6
1-0	Louapre, David	XXX	2900	2900	False	False	g6
0-1	Nepomniachtchi, Ian	Louapre, David	2900	2900	False	False	g6
0-1	Nepomniachtchi, Ian	Louapre, David	2000	2000	False	False	g6
0-1	Louapre, David	XXX	1400	1400	False	False	Nf6
1-0	XXX	Nepomniachtchi, Ian	1800	1800	False	False	g6
1-0	Nepomniachtchi, Ian	Louapre, David	2900	2900	False	False	g6
1-0	Louapre, David	Nepomniachtchi, Ian	1800	1800	False	False	g6
0-1	Louapre, David	Nepomniachtchi, Ian	1000	1000	False	False	g6
1-0	Nepomniachtchi, Ian	XXX	2000	2000	False	False	g6
0-1	XXX	Nepomniachtchi, Ian	2000	2000	False	False	g6
0-1	Nepomniachtchi, Ian	XXX	1700	1700	False	False	g6
1-0	Louapre, David	XXX	1700	1700	False	False	Nf6



MrPhi @MonsieurPhi
 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).



11:55 AM · Apr 19, 2024 · 447 Views
 Post your reply

MrPhi @MonsieurPhi · Apr 19
 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.

Solution #5: Strategic exploration with modelling and learning

```

original GM titles
:title = "[BlackTitle \"GM\"]" if include_black_title else ""
:title = "[WhiteTitle \"GM\"]" if include_white_title else ""
  
```

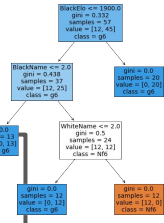
```

# Construct the PGN header with the configurable options
pgn_headers = f"""[Event "FIDE World Championship Match 2024"]
[Site "Los Angeles, USA"]
[Date "2024.12.01"]
[Round "5"]
[White "{white_name}"]
[Black "{black_name}"]
  
```

```

gpt_config = GPTConfig
model_gpt="gpt-3.5"
temperature=0.0,
max_tokens=5,
chat_gpt=False,
system_role_message=None # Since it wasn't provided in the original call
)

# 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]
  
```



Result	WhiteName	BlackName	WhiteElo	BlackElo	IncludeWhiteTitle	IncludeBlackTitle	Move
0-1	XXX	Louapre, David	1800	1800	False	False	Nf6
1-0	Louapre, David	XXX	2900	2900	False	False	g6
0-1	Nepomniachtchi, Ian	Louapre, David	2900	2900	False	False	g6
0-1	Nepomniachtchi, Ian	Louapre, David	2000	2000	False	False	g6
0-1	Louapre, David	XXX	1400	1400	False	False	Nf6
1-0	XXX	Nepomniachtchi, Ian	1800	1800	False	False	g6
1-0	Nepomniachtchi, Ian	Louapre, David	2900	2900	False	False	g6
1-0	Louapre, David	Nepomniachtchi, Ian	1800	1800	False	False	g6
0-1	Louapre, David	Nepomniachtchi, Ian	1000	1000	False	False	g6
1-0	Nepomniachtchi, Ian	XXX	2000	2000	False	False	g6
0-1	XXX	Nepomniachtchi, Ian	2000	2000	False	False	g6
0-1	Nepomniachtchi, Ian	XXX	1700	1700	False	False	g6
1-0	Louapre, David	XXX	1700	1700	False	False	Nf6



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
 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
 lee218@llnl.gov
 jones19@llnl.gov
 Lawrence Livermore National Laboratory

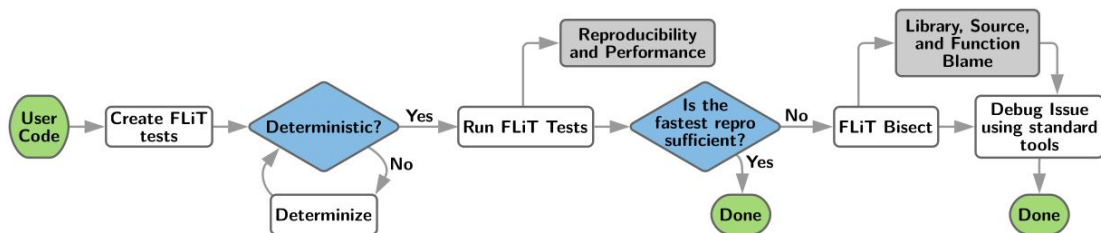


Figure 1: Multi-level workflow. Levels are (1) determine variability-inducing compilations, (2) analyze the space of reproducibility and performance, and (3) debug variability by identifying files and functions causing variability.

Flag	GCC	Clang	Intel	NVCC
-fassociative-math	X	X		
-fcx-fortran-rules	X			
-fcx-limited-range	X		X	
-fexcess-precision=fast	X	X		
-fexcess-precision=standard		X		
-ffinite-math-only	X	X		
-ffloat-store	X	X	X	
-ffp-contract=on	X	X		
-fma				X
-fmerge-all-constants	X	X	X	
-fno-trapping-math	X	X		
-fp-model fast=1				X
-fp-model fast=2				X
-fp-model=double				X
-fp-model=extended				X
-fp-model=precise				X
-fp-model=source				X
-fp-model=strict				X
-fp-port				X
-freciprocal-math	X	X		
-frounding-math	X	X	X	
-fsignaling-nans	X	X		
-fsingle-precision-constant			X	X
-ftz				X
-funsafe-math-optimizations	X	X		
-march=core-avx2			X	X
-mavx	X	X	X	
-mavx2 -mfma	X	X	X	
-mfpmath=sse -mtune=native	X	X	X	
-mpl				X
-no-fma				X
-no-ftz				X
-no-prec-div				X
--prec-div				X
--fmad=false				X
--fmad=true				X
--ftz=true				X
--prec-div=false				X
--prec-div=true				X
--prec-sqrt=false				X
--prec-sqrt=true				X
--use_fast_math			X	X

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.

```
#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 = %a\n", z);
    return 0;
}
```



```
#include <stdio.h>

// Define the type to use as an alias
#ifdef USE_DOUBLE
    typedef double real;
#elif defined(USE_FLOAT)
    typedef float real;
#else
    typedef long double real;
#endif

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

    // Print the value of z
#ifdef USE_DOUBLE
    printf("z (using double) = %a\n", z);
#elif defined(USE_FLOAT)
    printf("z (using float) = %a\n", z);
#else
    printf("z (using long double) = %La\n", z);
#endif

    return 0;
}
```

Solution #7: LLMs to support exploration of variants space

```
#include <stdio.h>

// Define the type to use as an alias
#ifdef USE_DOUBLE
    typedef double real;
#elif defined(USE_FLOAT)
    typedef float real;
#else
    typedef long double real;
#endif

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

    // Print the value of z
#ifdef USE_DOUBLE
    printf("z (using double) = %a\n", z);
#elif defined(USE_FLOAT)
    printf("z (using float) = %a\n", z);
#else
    printf("z (using long double) = %La\n", z);
#endif

    return 0;
}
```

Write a Python script that compiles and 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]},
        ])

    return data
```

```
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]},
        ])

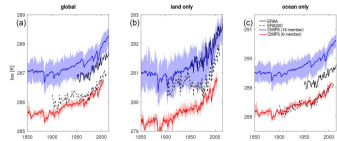
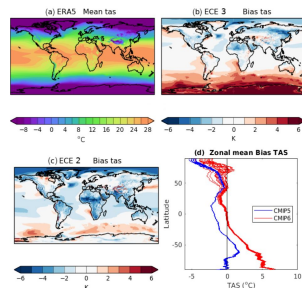
```

compiler	flags	type	result
gcc		original	z (using long double) = 0x8.0000000000004p-3
gcc		double	z (using double) = 0x1.0000000000001p+0
gcc		float	z (using float) = 0x1p+0
gcc	-mfpmath=387	original	z (using long double) = 0x8.0000000000004p-3
gcc	-mfpmath=387	double	z (using double) = 0x1p+0
gcc	-mfpmath=387	float	z (using float) = 0x1p+0
gcc	-ffast-math	original	z (using long double) = 0x8.0000000000004p-3
gcc	-ffast-math	double	z (using double) = 0x1.0000000000001p+0
gcc	-ffast-math	float	z (using float) = 0x1p+0
gcc	-fno-signed-zeros	original	z (using long double) = 0x8.0000000000004p-3
gcc	-fno-signed-zeros	double	z (using double) = 0x1.0000000000001p+0
gcc	-fno-signed-zeros	float	z (using float) = 0x1p+0
gcc	-fno-trapping-math	original	z (using long double) = 0x8.0000000000004p-3
gcc	-fno-trapping-math	double	z (using double) = 0x1.0000000000001p+0
gcc	-fno-trapping-math	float	z (using float) = 0x1p+0
gcc	-fno-rounding-math	original	z (using long double) = 0x8.0000000000004p-3
gcc	-fno-rounding-math	double	z (using double) = 0x1.0000000000001p+0
gcc	-fno-rounding-math	float	z (using float) = 0x1p+0
clang		original	z (using long double) = 0x8.0000000000004p-3
clang		double	z (using double) = 0x1.0000000000001p+0

Retrieve the result of S. Boldo et al.

- ▶ **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



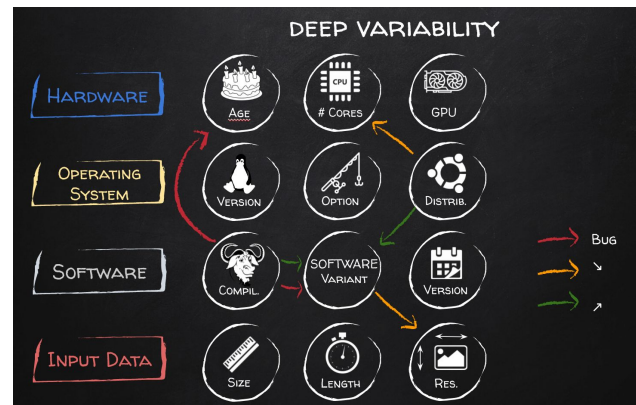
- ▶ **Replicability is the holy grail!**

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

- ▶ **Some solutions**

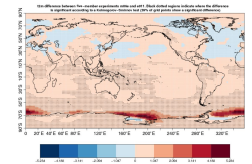
- ▶ abstractions/models
- ▶ learning and sampling
- ▶ reuse of configuration knowledge
- ▶ leveraging stability
- ▶ systematic exploration
- ▶ identification of root causes
- ▶ LLMs to support exploration of variants' space
- ▶ incremental build of configuration space (Randrianaina et al. ICSE'22)
- ▶ 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**



Backup slides (disclaimer: don't try to understand everything ;))

What can we do? (robustness)



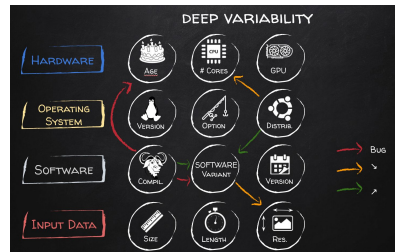
Robustness (trustworthiness) of scientific results to sources of variability

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



different data

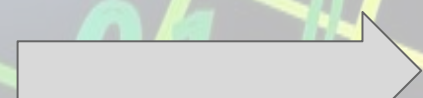


deep

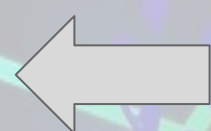
software

variability

different methods



different assumptions



different analyses



Variability in the analysis of a single neuroimaging dataset by many teams

Rotem Botvink-Nezer, Felix Holzmeister, ... Tom Schonberg + Show authors

Nature 582, 84–88 (2020) | [Cite this article](#)

42k Accesses | 203 Citations | 2056 Altmetric | [Metrics](#)

Increasing Transparency Through a Multiverse Analysis

Sara Steegen ¹, Francis Tuerlinckx ¹, Andrew Gelman ², Wolf Vanpaemel ³

Affiliations + expand

PMID: 27694465 DOI: 10.1177/1745691616658637

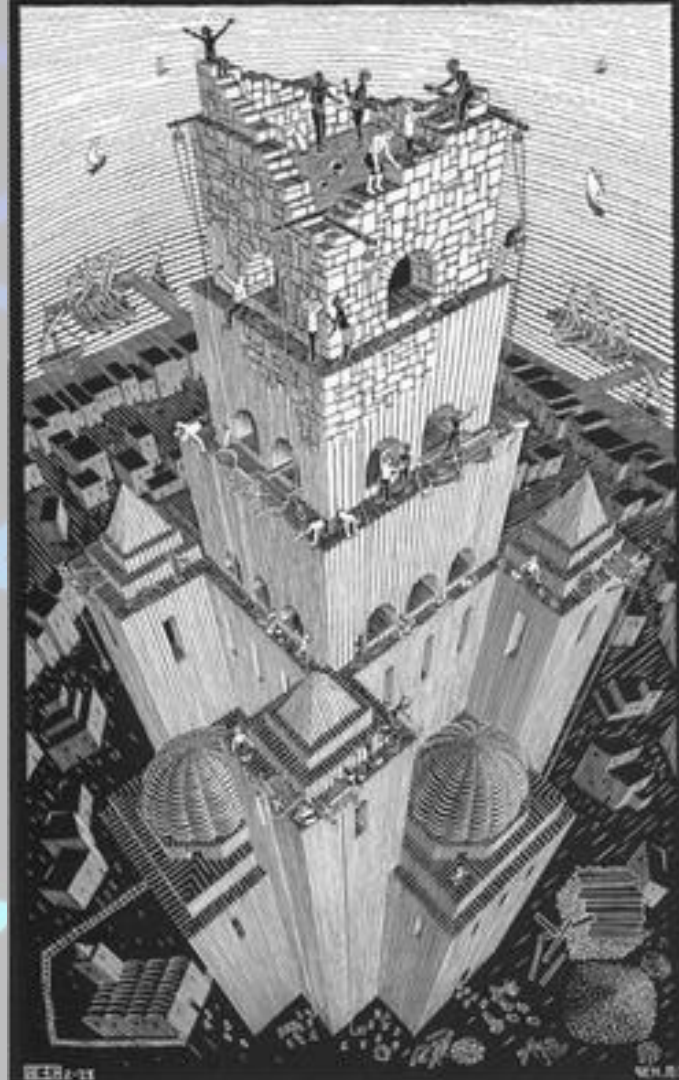
Many Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results

R. Silberzahn, E. L. Uhlmann, D. P. Martin, more... [Show all authors](#)

First Published August 23, 2018 | Research Article



<https://doi.org/10.1177/2515245917747646>



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 vs replication <http://rescience.github.io/faq/>

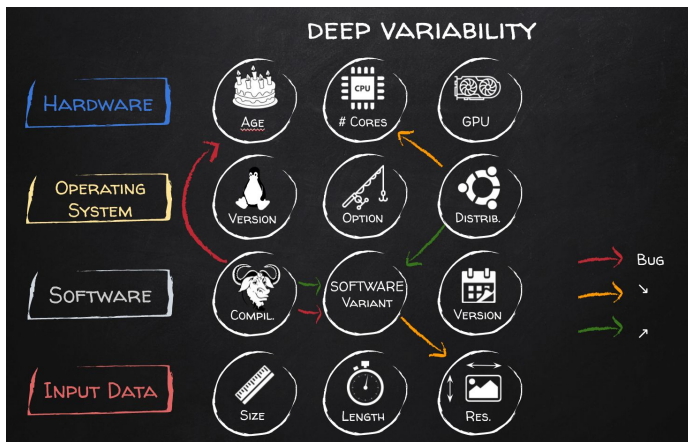
“**Reproduction** of a computational study means running the same computation on the same input data, and then checking if the results are the same, or at least “close enough” when it comes to numerical approximations. Reproduction can be considered as **software testing** at the level of a complete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

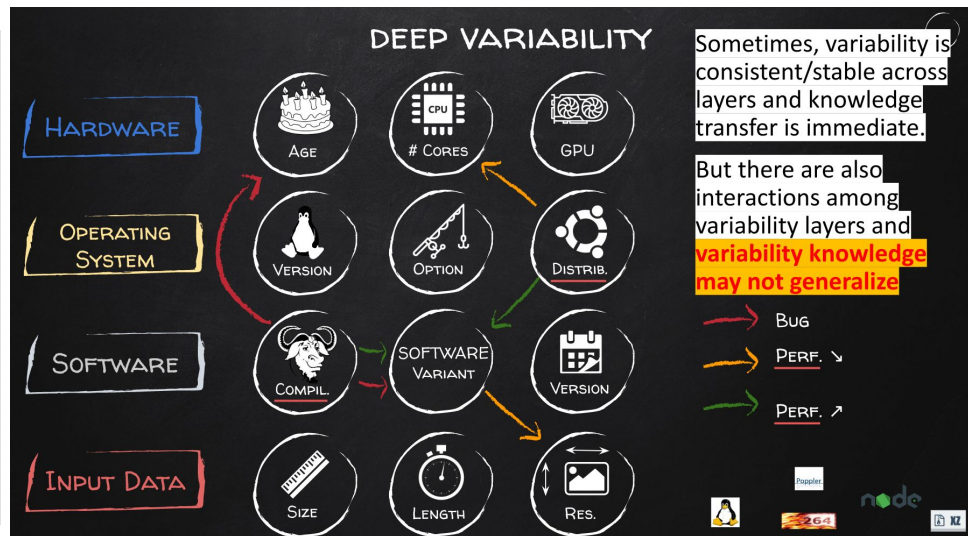
"Terminologies for Reproducible Research", Lorena A. Barba, 2018



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.

A	B1	B2
political science economics	signal processing scientific computing econometrics epidemiology clinical studies internal medicine physiology (neuro) computational biology biomedical research statistics	microbiology, immunology (FASEB) computer science (ACM)





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.

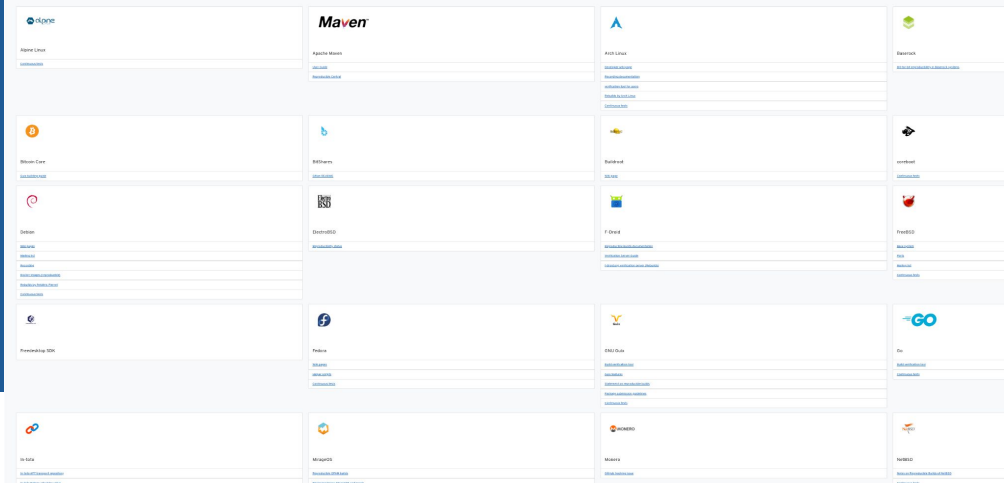
This is particularly a concern for developers collaborating on privacy or security software: attacking these typically result in compromising particularly politically-sensitive targets such as dissidents, journalists and whistleblowers, as well as anyone wishing to communicate securely under a repressive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



Transferring Performance Prediction Models Across Different Hardware Platforms

Valov et al. ICPE 2017

Table 3: Summary of measured systems; N_f – Number of features; NM – Number of machines on which systems were measured; NMC – Number of measured configurations

System	N_f	NM	NMC
XZ	7	7	154
x264	7	11	165
SQLite	5	10	32

Table 2: Summary of hardware platforms on which configurable software systems were measured; MID – Machine ID in DataMill cluster; NC – Number of CPUs; IS – Instruction set; CCR – CPU clock rate (MHz); RAM – RAM memory size (MB)

Systems			Machines				
XZ	x264	SQLite	MID	NC	IS	CCR	RAM
✓			73	2	i686	1733	1771
✓	✓	✓	75	2	i686	3200	977
✓			77	2	i686	2992	2024
✓			78	1	i686	1495	755
✓			79	4	x86_64	3291	7961
✓			80	8	x86_64	3401	7907
✓	✓		81	16	x86_64	2411	32193
	✓		87	1	i686	1595	249
	✓		88	1	i686	1700	978
		✓	90	2	i686	3200	977
	✓		91	1	i686	2400	1009
	✓		97	2	i686	2992	873
	✓	✓	98	2	i686	2992	873
		✓	99	2	i686	2793	880
			103	2	i686	3200	881
			104	1	i686	1800	502
		✓	105	2	i686	3200	881
	✓		106	2	i686	3192	494
		✓	125	4	x86_64	3301	7960
	✓		128	2	i686	2993	2024
		✓	130	2	i686	3198	880
		✓	146	2	i686	2998	872
		✓	157	36	x86_64	2301	15954

“**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

SPEAR (SAT Solver)	X264 (video encoder)	SQLite (DB engine)	SaC (Compiler)
Analysis time	Encoding time	Query time	Execution time
14 options	16 options	14 options	50 options
16,384 configurations	4,000 configurations	1,000 configurations	71,267 configurations
SAT problems	Video quality/size	DB Queries	10 Demo programs
3 hardware	2 hardware	2 hardware	
2 versions	3 versions	2 versions	

$ec_1 : [h_2 \rightarrow h_1, w_3, v_3]$	SM	0.97
$ec_2 : [h_2 \rightarrow h_1, w_1, v_3]$	S	0.96
$ec_3 : [h_1, w_1 \rightarrow w_2, v_3]$	M	0.65
$ec_4 : [h_1, w_1 \rightarrow w_3, v_3]$	M	0.67
$ec_5 : [h_1, w_3, v_2 \rightarrow v_3]$	L	0.05
$ec_6 : [h_1, w_3, v_1 \rightarrow v_3]$	L	0.06
$ec_7 : [h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3]$	L	0.08
$ec_8 : [h_2 \rightarrow h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3]$	VL	0.09



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.

Transfer Learning for Software Performance Analysis: An Exploratory Analysis

Jamshidi et al. ASE 2017

SPEAR (SAT Solver)	X264 (video encoder)	SQLite (DB engine)	SaC (Compiler)
Analysis time	Encoding time	Query time	Execution time
14 options	16 options	14 options	50 options
16,384 configurations	4,000 configurations	1,000 configurations	71,267 configurations
SAT problems	Video quality/size	DB Queries	10 Demo programs
3 hardware	2 hardware	2 hardware	
2 versions	3 versions	2 versions	

$ec_1 : [h_2 \rightarrow h_1, w_3, v_3]$	SM	0.97
$ec_2 : [h_2 \rightarrow h_1, w_1, v_3]$	S	0.96
$ec_3 : [h_1, w_1 \rightarrow w_2, v_3]$	M	0.65
$ec_4 : [h_1, w_1 \rightarrow w_3, v_3]$	M	0.67
$ec_5 : [h_1, w_3, v_2 \rightarrow v_3]$	L	0.05
$ec_6 : [h_1, w_3, v_1 \rightarrow v_3]$	L	0.06
$ec_7 : [h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3]$	L	0.08
$ec_8 : [h_2 \rightarrow h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3]$	VL	0.09



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!

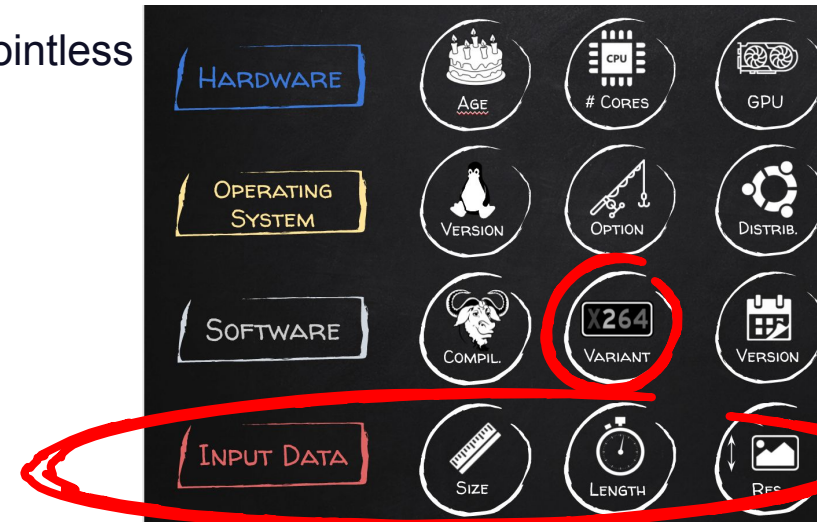
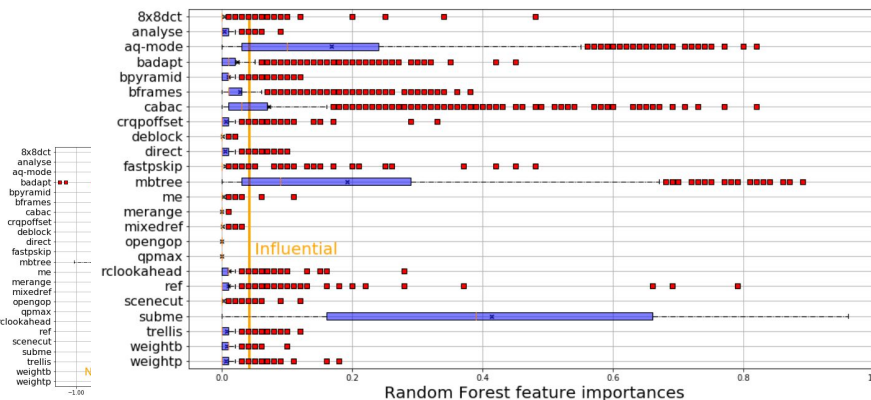
Practical impacts for users, developers, scientists, and self-adaptive systems

Threats to variability knowledge for performance property bitrate



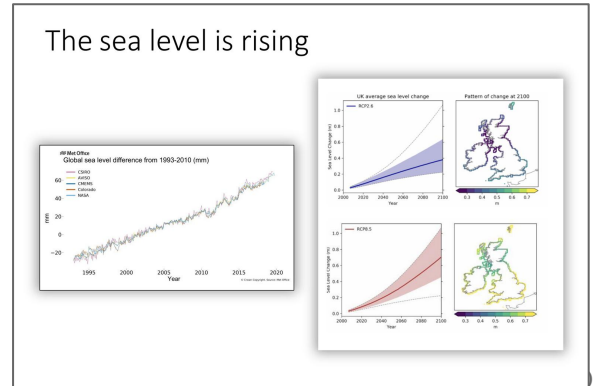
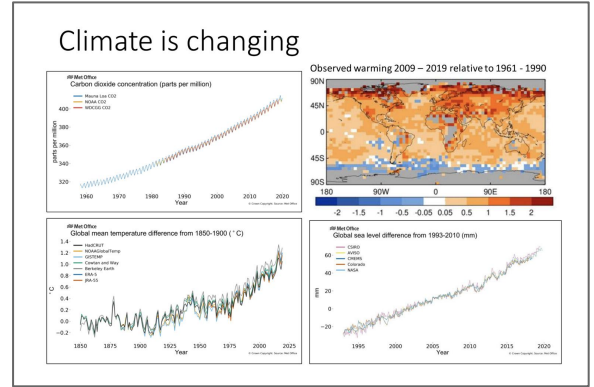
- optimal configuration is specific to an input; a good configuration can be a bad one
- some options' values have an opposite effect depending on the input
- effectiveness of sampling strategies (random, 2-wise, etc.) is input specific (somehow confirming Pereira et al. ICPE 2020)
- predicting, tuning, 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

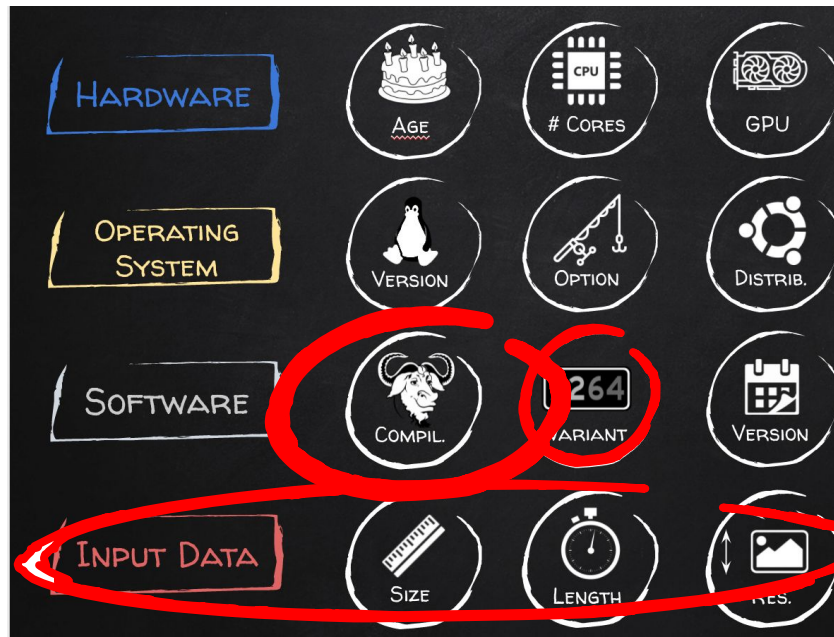
--disable-thread disable multithreaded encoding
--disable-win32thread disable win32threads (windows only)
--disable-interlaced disable interlaced encoding support
--bit-depth=BIT_DEPTH set output bit depth (8, 10, all) [all]
--chroma-format=FORMAT output chroma format (400, 420, 422, 444, all) [all]

Advanced options:
--disable-asm disable platform-specific assembly optimizations
--enable-lto enable link-time optimization
--enable-debug add -g
--enable-gprof add -pg
--enable-strip add -s
--enable-pic 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-avs disable avisynth support
--disable-awscale disable swscale support
--disable-lavf disable libavformat support
--disable-ffms disable ffmpegsource support
--disable-gpac disable gpac support
--disable-lsmash disable lsmash support

compile-time options



What can we do? (#1 studies)

Empirical studies about deep software variability

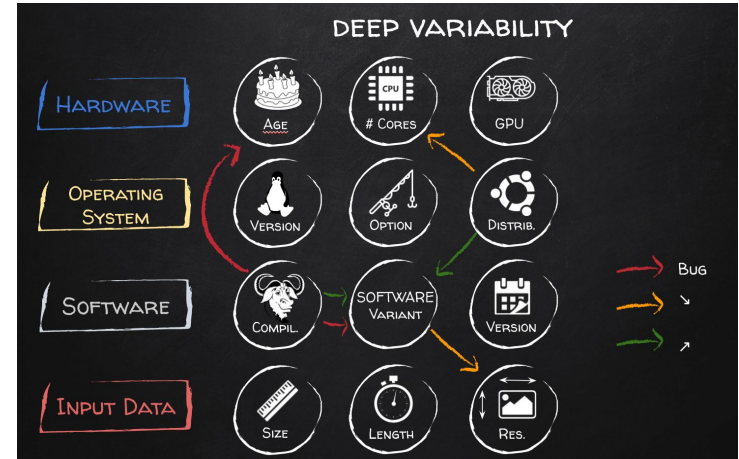
- more subject systems
- more variability layers, including interactions
- more quantitative (e.g., performance) properties

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**

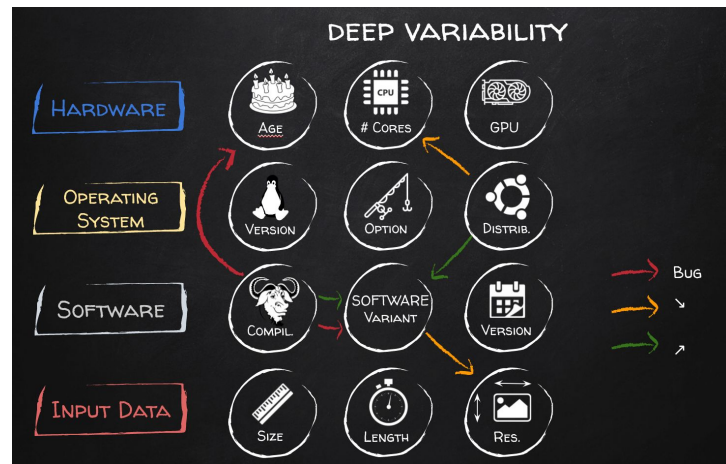


What can we do? (#2 cost)

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
- ...



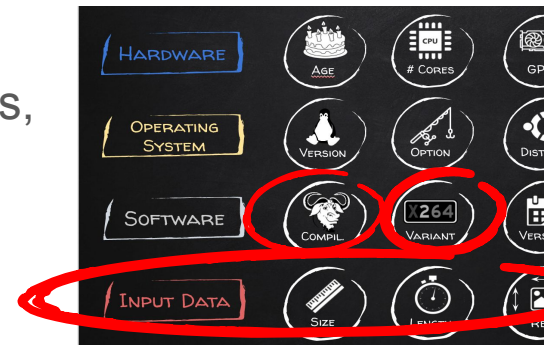
Key results (for x264)

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.



Our Vision

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

<https://hal.science/hal-04582287>

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

Georges Aaron Randrianaina
Univ Rennes, Inria, CNRS, IRISA
Rennes, France

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

Jean-Marc Jézéquel
Univ Rennes, Inria, CNRS, IRISA, IUF
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 parties 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 and vision paper we delve into the application of software engineering techniques, specifically variability management, to systematically identify and explicit 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.

1 INTRODUCTION

Many scientific domains need to process large amount of data with more and more complex computations. For instance, studies about climate modelling and change involve the design of mathematical model, the mining and analysis of data, the executions of large simulations, etc. [16, 24, 29]. 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 overstatement to say that computational science depends on software and its engineering [2, 34, 54].

One of the main promise of software is that a result obtained by an experiment (e.g., a simulation) can 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, 15, 19, 22, 31, 41, 42, 53]. For instance, applications of different analysis pipelines, alterations

community [22]. As a result, software can threaten the scientific knowledge and recommendations built on top of these computations and studies.

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 hypotheses.

2 DEEP SOFTWARE VARIABILITY

Uncertainty in informatics comes from many different origins [17, 39], either ontological (*i.e.*, inherent unpredictability, e.g., aleatory) 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 processor¹.

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., $-3\%2$ evaluates to -1 in Java but to 1 in Python), or even inside the same programming language (for instance $x/0$ is an undefined behavior in C).

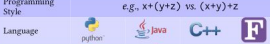


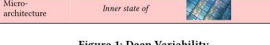


Parameters:	e.g., random seed selection
Input Data	
Programming Style	e.g., $x+(y+z)$ vs. $(x+y)+z$
Language	
Compiler & VM	
Library	
Platform	
Processor	
Micro-architecture	Inner state of 

Figure 1: Deep Variability

⇒ 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”.

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
 lee218@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.

```

-fcx-fortran-rules X
-fcx-limited-range X X
-fcxexes-precision-fast X X
-fcxexes-precision-standard X X
-ffloat-store X X X
-ffp-contract=on X X
-fma X X X
-fmerge-all-constants X X X
-fno-trapping-math X X
-fp-model fast=1 X
-fp-model fast=2 X
-fp-model=module X
-fp-model=extended X
-fp-model=precise X
-fp-model=source X
-fp-model=strict X
-fp-prec X
-ffp-prec=prcal-math X X
-frounding-math X X X
-fsignaling-mans X X
-fsingle-precision-constant X X
-ftz X X X
-funsafe-math-optimizations X X X
-march=core-avx2 X X X
-maxx X X X
-maxx2 -fma X X X
-mfmath=ase -m tune=native X X X
-mpi X
-no-fma X
-no-ftz X
-no-prec-div X
-prec-div X
-fmad=false X
-fmad=true X
-ftz=true X
-prec-div=false X
-prec-div=true X
-prec-sqrt=false X
-prec-sqrt=true X
-use_fast_math X X
    
```

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.

Compiler	Released	# Variable Runs	Best Flags	Speedup
gcc-8.2.0	26 July 2018	78 of 1,288 (6.0%)	-O2 -funsafe-math-optimizations	1.097
clang-6.0.1	05 July 2018	24 of 1,368 (1.8%)	-O3 -funsafe-math-optimizations	1.042
icpc-18.0.3	16 May 2018	984 of 1,976 (49.8%)	-O2 -fp-model fast=2	1.056

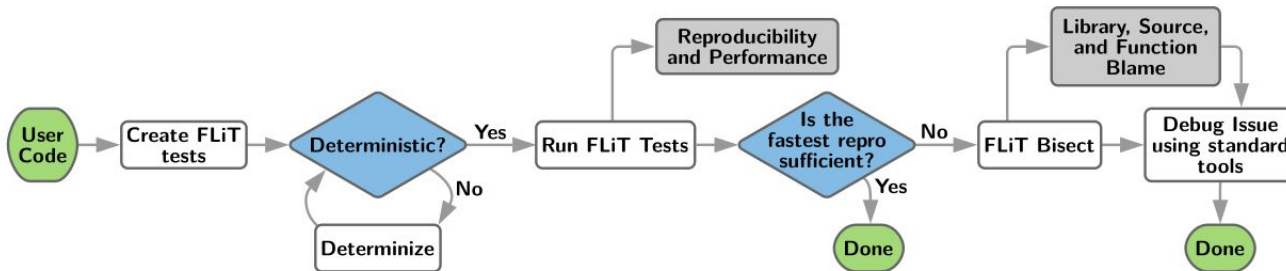


Figure 1: Multi-level workflow. Levels are (1) determine variability-inducing compilations, (2) analyze the space of reproducibility and performance, and (3) debug variability by identifying files and functions causing variability.

$\text{exec}(\text{software}) = \text{exec_repro}(\text{software})$

or

$\text{exec}(\text{software}) \sim= \text{exec}(\text{software_repro})$

(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)

$\text{exec}(\text{software}) \neq \text{exec_repro}(\text{software})$

$\text{software} \sim= \text{software_repro}$

$\text{exec}(\text{software}, \text{hardware})$

$\text{exec}(\text{software}, \text{hardware}, \text{compiler}, \text{input_data}, \text{operating_system}, \text{bios}, \text{container}, \text{hypervisor}, \text{dependencies_versions})$

$\text{exec}(v_1, v_2, \dots, v_N) \sim= \text{exec_repro}(v_1', v_2', \dots, v_{N'})$

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)

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
 lee218@llnl.gov
 jones19@llnl.gov
 Lawrence Livermore National Laboratory

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.

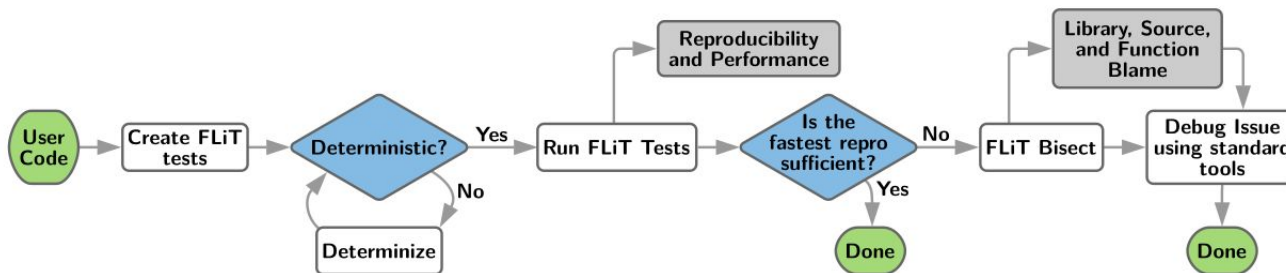


Table 1: Compilers used in the MFEM study with summary statistics. The best flags are chosen by the best average speedup across all MFEM examples. The average speedup over all 19 MFEM examples is reported and is calculated relative to the speedup of g++ -O2.

Compiler	Released	# Variable Runs	Best Flags	Speedup
gcc-8.2.0	26 July 2018	78 of 1,288 (6.0%)	-O2 -funsafe-math-optimizations	1.097
clang-6.0.1	05 July 2018	24 of 1,368 (1.8%)	-O3 -funsafe-math-optimizations	1.042
icpe-18.0.3	16 May 2018	984 of 1,976 (49.8%)	-O2 -fp-model fast=2	1.056

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 (NeurIPs 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>

<https://statmodeling.stat.columbia.edu/2023/10/13/frictionless-reproducibility-methods-as-protocols-division-of-labor-as-a-characteristic-of-statistical-methods-statistics-as-the-science-of-defaults-statisticians-well-prepared-to-think-about/>

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://hdr.mitpress.mit.edu/pub/8dqqwqiu/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/8dqqwqiu/release/1>

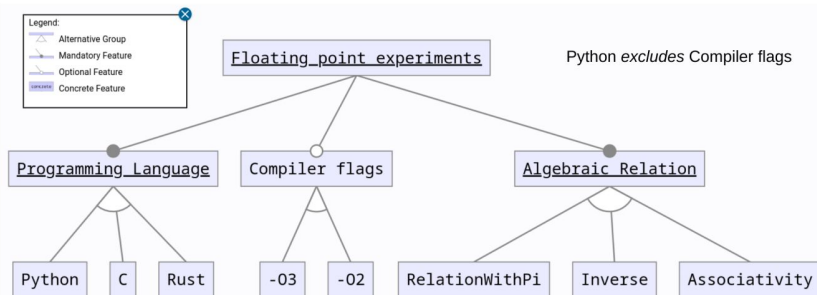


Figure 2: Feature model (excerpt). Inverse (resp. RelationWithPi) corresponds to checking the property $(x * z) / (y * z) = x / y$ (resp. $(x * z * \pi) / (y * z * \pi) = x / y$) with $z, y \neq 0$

```

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)
  
```

```

fn check_ratio(config: &Config, x: f64, y: f64, z: f64) -> bool {
    if let Some(error_margin) = config.error_margin {
        #[cfg(feature = "associativity")]
        {
            ((x + y) + z - x - (y + z)).abs() < error_margin
        }
        #[cfg(feature = "mult_inverse")]
        {
            ((x * z) / (y * z) - x / y).abs() < error_margin
        }
        #[cfg(feature = "mult_inverse_pi")]
        {
            ((x * z * std::f64::consts::PI) / (y * z * std::f64::consts::PI) - x / y).abs() < error_margin
        }
    } else {
        #[cfg(feature = "associativity")]
        {
            (x + y) + z == x + (y + z)
        }
        #[cfg(feature = "mult_inverse")]
        {
            (x * z) / (y * z) == x / y
        }
        #[cfg(feature = "mult_inverse_pi")]
        {
            (x * z * std::f64::consts::PI) / (y * z * std::f64::consts::PI) == (x / y)
        }
    }
}
  
```

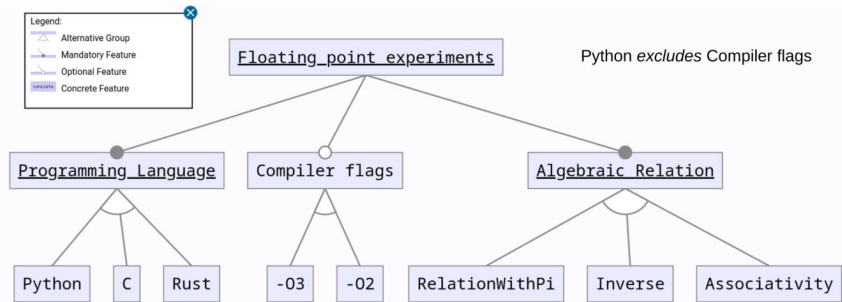



Figure 2: Feature model (excerpt). Inverse (resp. Relation-WithPi) corresponds to checking the property $(x * z) / (y * z) = x / y$ (resp. $(x * z * \pi) / (y * z * \pi) = x / y$) with $z, y \neq 0$

```

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 {
    StdRng::seed_from_u64(seed_val);
    let mut ok = 0;
    for _ in 0..number {
        if associativity_test(config) {
            ok += 1;
        }
    }
    ok * 100 / number
}
  
```

```

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

```

fn check_ratio(config: &Config, x: f64, y: f64, z: f64) -> bool {
    if let Some(error_margin) = config.error_margin {
        #[cfg(feature = "associativity")]
        {
            ((x + y) + z - x - (y + z)).abs() < error_margin
        }
        #[cfg(feature = "mult_inverse")]
        {
            ((x * z) / (y * z) - x / y).abs() < error_margin
        }
        #[cfg(feature = "mult_inverse_pi")]
        {
            ((x * z * std::f64::consts::PI) / (y * z * std::f64::consts::PI) - x / y).abs() < error_margin
        }
    } else {
        #[cfg(feature = "associativity")]
        {
            (x + y) + z == x + (y + z)
        }
        #[cfg(feature = "mult_inverse")]
        {
            (x * z) / (y * z) == x / y
        }
        #[cfg(feature = "mult_inverse_pi")]
        {
            (x * z * std::f64::consts::PI) / (y * z * std::f64::consts::PI) == (x / y)
        }
    }
}
  
```

Language	Library	System	Compiler	VariabilityMisc	EqualityCheck	NumberGenerations	Repeat	min	max	std	mean
Perl				seed None	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
Perl				seed None	MULT_INV	100	10	60.0	71.0	3.562302626111375	65.1
Perl				seed None	MULT_INV_PI	100	10	51.0	63.0	3.330165161069343	55.9
Perl				seed 42	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
Perl				seed 42	MULT_INV	100	10	62.0	62.0	0.0	62.0
Perl				seed 42	MULT_INV_PI	100	10	47.0	47.0	0.0	47.0
Go				seed None	associativity	100	10	71.0	82.0	3.3466401061363023	76.0
Go				seed None	mult-inverse	100	10	58.0	78.0	6.0	66.0
Go				seed None	mult-inverse-pi	100	10	42.0	64.0	5.885575587824865	53.4
Go				seed 42	associativity	100	10	81.0	81.0	0.0	81.0
Go				seed 42	mult-inverse	100	10	70.0	70.0	0.0	70.0
Go				seed 42	mult-inverse-pi	100	10	56.0	56.0	0.0	56.0
R				seed None	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
R				seed None	MULT_INV	100	10	62.0	72.0	2.764054992217051	66.6
R				seed None	MULT_INV_PI	100	10	47.0	57.0	2.808914381037628	53.1
R				seed 42	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
R				seed 42	MULT_INV	100	10	67.0	67.0	0.0	67.0
R				seed 42	MULT_INV_PI	100	10	53.0	53.0	0.0	53.0
Julia				seed None strict-equality	ASSOCIATIVITY	100	10	74.0	90.0	4.6097722286464435	82.5
Julia				seed None strict-equality	MULT_INV	100	10	60.0	79.0	6.16765757804371	68.6
Julia				seed None strict-equality	MULT_INV_PI	100	10	49.0	59.0	2.8301943396169813	54.3
Julia				seed None approximate equality	ASSOCIATIVITY	100	10	89.0	89.0	0.0	89.0
Julia				seed 42 strict-equality	MULT_INV	100	10	73.0	73.0	0.0	73.0
Julia				seed 42 strict-equality	MULT_INV_PI	100	10	55.0	55.0	0.0	55.0
Julia				seed None approximate equality of Julia lang	ASSOCIATIVITY	100	10	100.0	100.0	0.0	100.0
Julia				seed None approximate equality of Julia lang	MULT_INV	100	10	100.0	100.0	0.0	100.0

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