

# Reproducible Science and Software Engineering

Mathieu Acher @acherm  
Paul Temple @pleupi22  
Olivier Barais @barais



Université  
de Rennes

**INSA**



institut  
universitaire  
de France



UMR

IRISA

# Reproductibilité des résultats et génie logiciel empirique

**Résumé:** L'une des principales promesses des logiciels est qu'un résultat obtenu par une expérience (par exemple, une simulation) puisse être reproduit avec un degré élevé de concordance. La recherche de la reproductibilité a un impact sur différents domaines scientifiques, prend différentes formes et nécessite de rendre toutes les données et le code disponibles de manière que les calculs puissent être exécutés à nouveau avec des résultats identiques. Dans ce cours, nous passerons d'abord en revue les terminologies (par exemple, reproductible vs réplication vs répétition), les outils de base (par exemple, les systèmes de version, les "build systems", les gestionnaires de paquets, les "notebooks"), et les techniques (par exemple, les tests automatisés, l'intégration et le déploiement continu, la gestion de la configuration) liés à la science reproductible.

Par la suite, nous programmerons une fonctionnalité relativement simple dans différentes variantes et démontrerons que de nombreux facteurs (y compris les langages de programmation, les versions de bibliothèques, les compilateurs, les types de variables, la gestion de l'aléatoire, etc.) peuvent avoir un impact sur le résultat final. Grâce à cet exercice, nous souhaitons présenter et discuter des techniques logicielles, des méthodologies et des outils que les développeurs ou les scientifiques peuvent utiliser pour aborder et atténuer les problèmes de reproductibilité, en espérant que cela conduise à des résultats plus robustes et plus généraux.

École Jeunes Chercheuses et Jeunes  
Chercheurs en Programmation (EJCP)

L'école du GDR GPL

# Reproducible Science and Software Engineering

**Abstract:** One of the main promises of software is that a result obtained from an experiment (e.g. a simulation) can be reproduced with a high degree of concordance. The quest for reproducibility has an impact on different scientific fields, takes different forms and requires making all the data and code available so that calculations can be run again with identical results. In this course, we will first review terminologies (e.g. reproducible vs replication vs repeatability), basic tools (e.g. versioning systems, build systems, package managers, notebooks), and techniques (e.g. automated testing, continuous integration and deployment, configuration management) related to reproducible science.

We will then program a relatively simple feature in different variants and demonstrate that many factors (including programming languages, library versions, compilers, variable types, randomness management, etc.) can have an impact on the final result. Through this exercise, we aim to present and discuss software techniques, methodologies and tools that developers or scientists can use to address and mitigate reproducibility issues, to obtain more robust and general results.

École Jeunes Chercheuses et Jeunes  
Chercheurs en Programmation (EJCP)

L'école du GDR GPL

# About us...

Mathieu Acher <https://www.mathieuacher.com/>

Professeur INSA Rennes

Thème de recherche principal : la variabilité logicielle et les systèmes hautement configurables comme le kernel Linux ou les chaînes de traitements scientifiques.

SE+AI



Paul Temple est maître de conférences à l'Université de Rennes. Il a obtenu sa thèse en 2018 en étudiant le problème de prédiction de performances des systèmes configurables tout en y ajoutant une dimension supplémentaire qui est l'ensemble d'entrées à fournir à ces systèmes. Postdoc à l'UNamur, Namur, Belgique sous la supervision de Gilles Perrouin et de Patrick Heymans.

Olivier Barais, Professeur à l'Université de Rennes, responsable de l'équipe DiverSE  
<https://olivier.barais.fr/>





# AGENDA

## **Reproducible Science and (Deep) Software (Variability)**

Deep Software Variability

Evidence of Deep Software Variability in Science

Threats and Opportunities

# AGENDA

**Interactive course: Reproducible Science and (Deep) Software (Variability)**

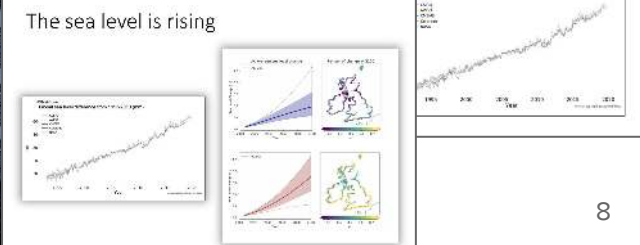
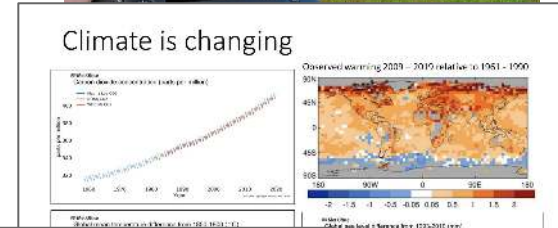
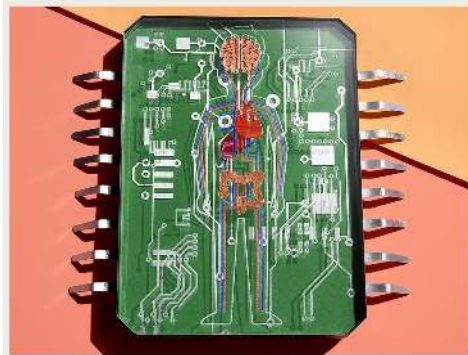
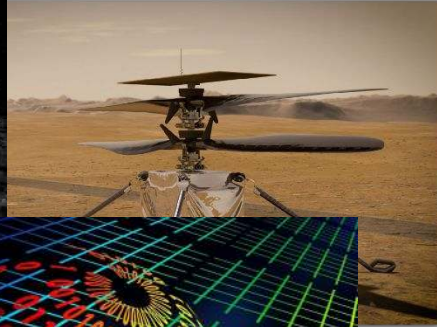
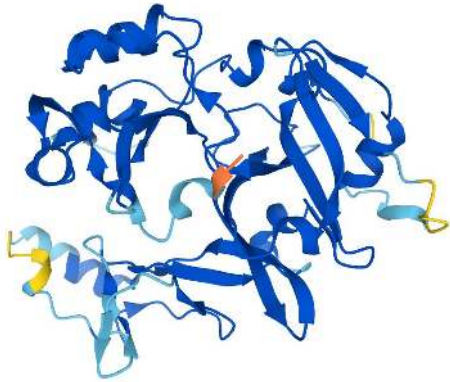
Is  $(x+y)+z == x+(y+z)$ ? (TP1)

A suspicious paper in machine learning... (TP2)

# SOFTWARE VARIANTS ARE EATING THE WORLD



# SOFTWARE IS EATING SCIENCE



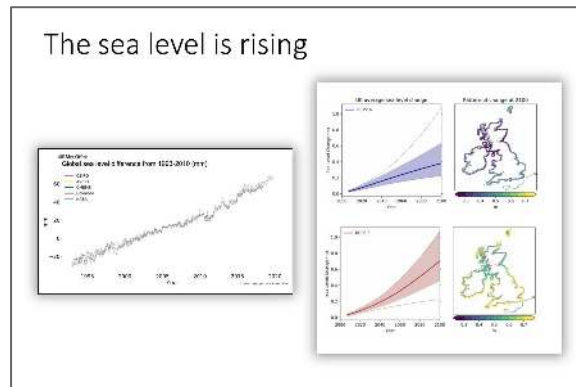
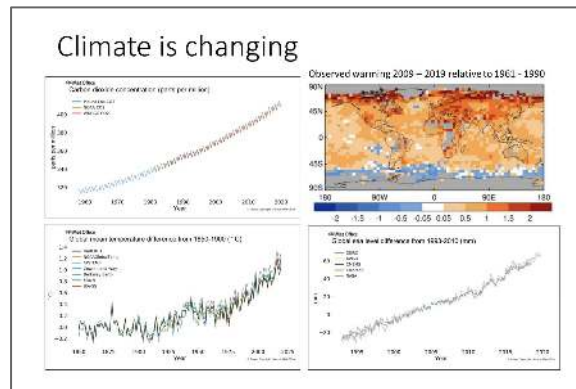


# Computational science depends on software and its engineering



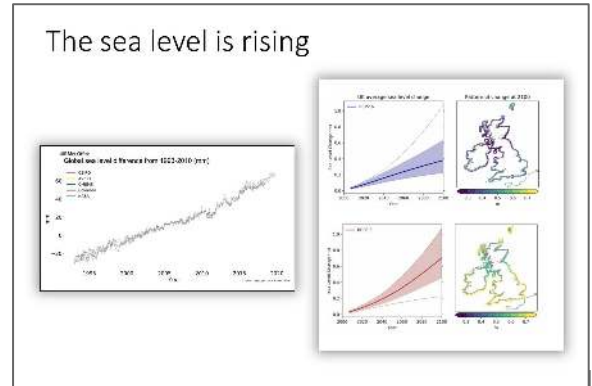
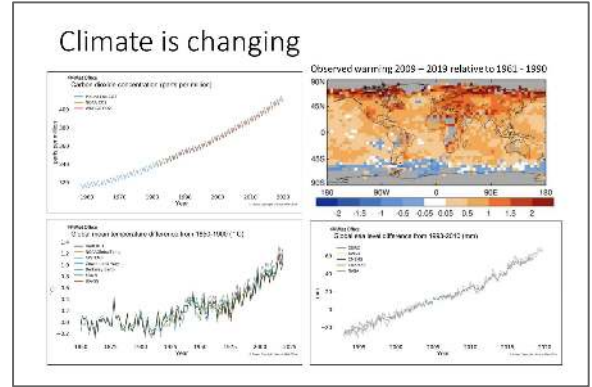
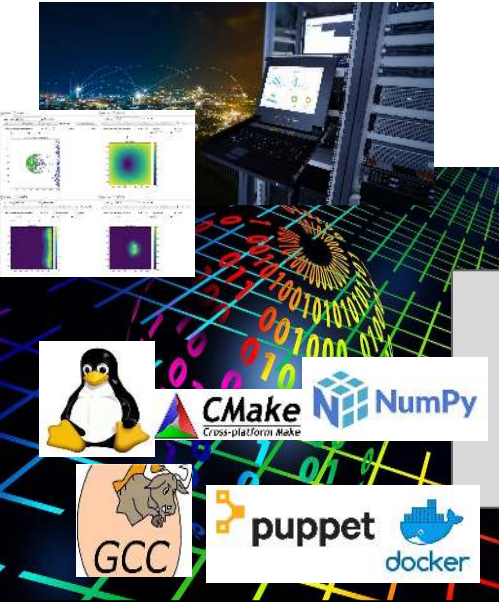
design of mathematical model  
mining and analysis of data  
executions of large simulations  
problem solving  
executable paper

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

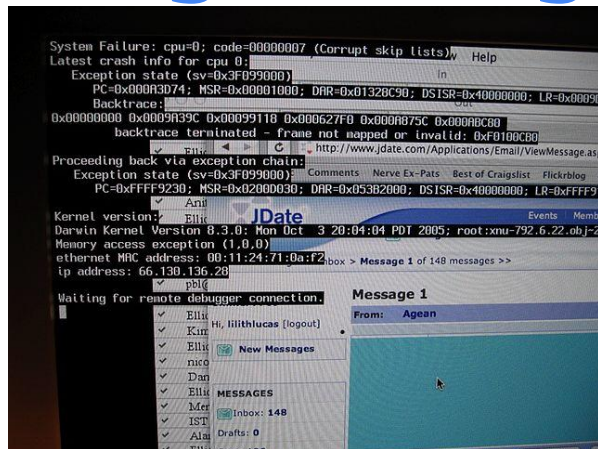
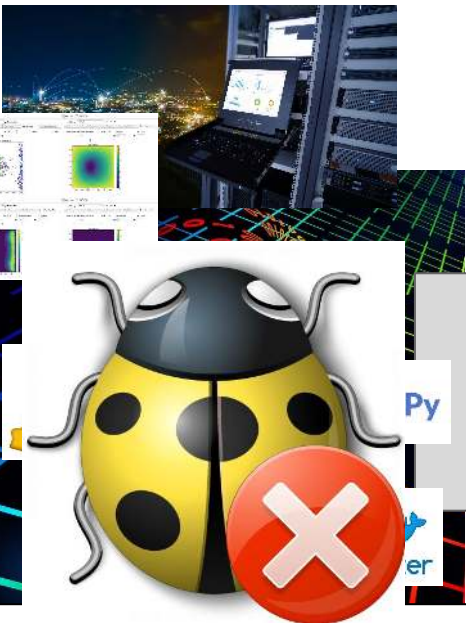
multi-million line of code base  
multi-dependencies  
multi-systems  
multi-layer  
multi-version  
multi-person  
multi-variant



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

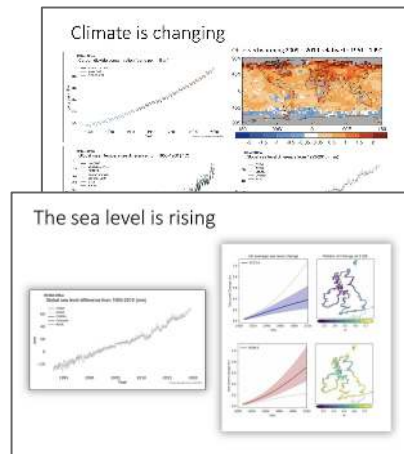
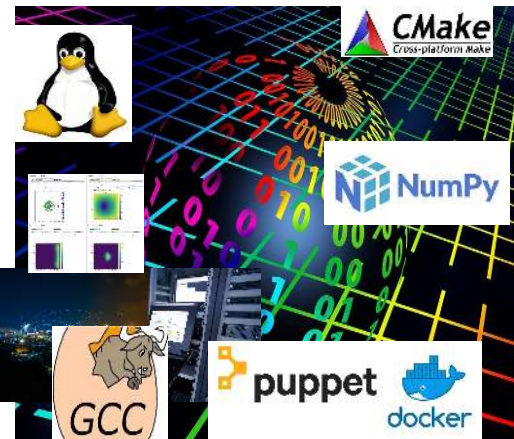
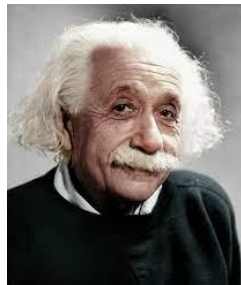


```
In [6]: sess.run(tf.svd(tf_matrix))
Out[6]: (array([[ 9.99998987e-01,  1.48747023e-03,  4.88133628e-06,
  4.69611084e-06,  4.37980998e-06,  3.45290823e-06,
  1.14686304e-06,  3.10980795e-06,  2.97625912e-06,
  2.65099743e-06,  1.91537106e-06,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  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, -2.61201495e-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>



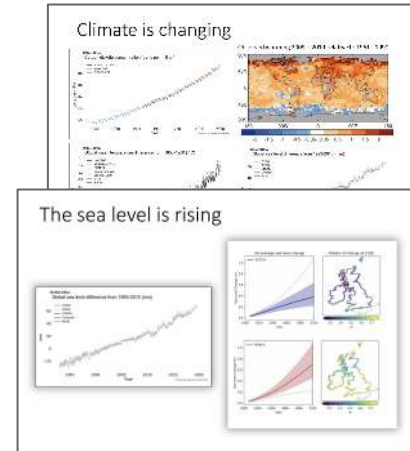
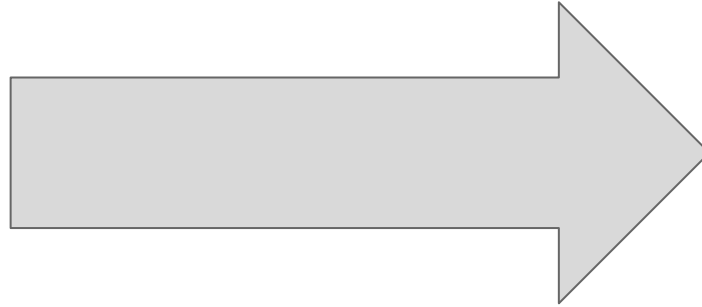
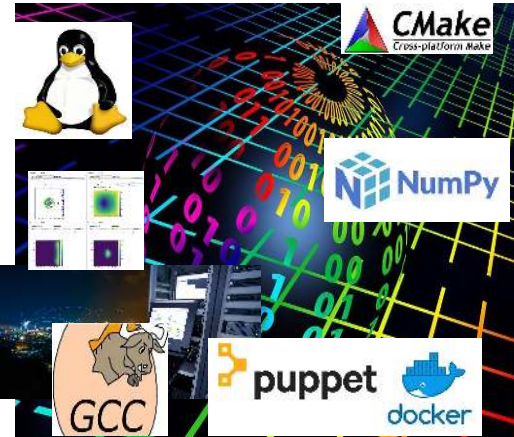


# Reproducibility

“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)

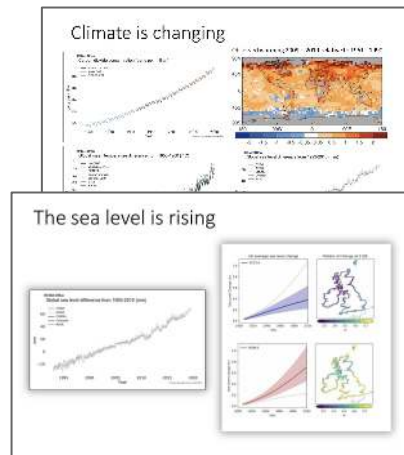
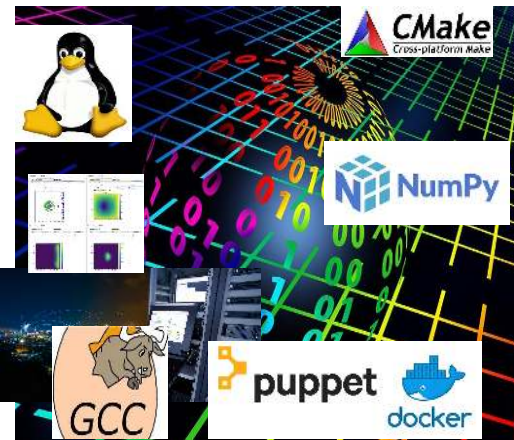


# Reproducibility and Replicability

**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

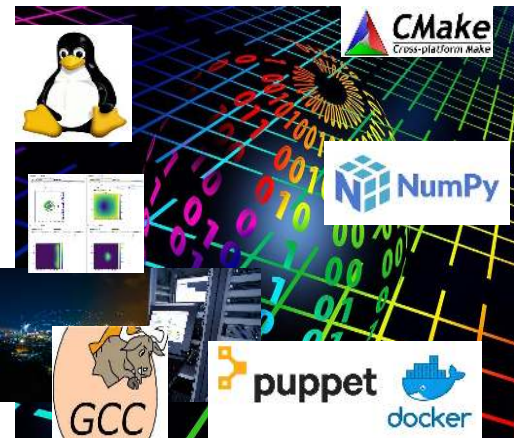


# Reproducibility and Replicability

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

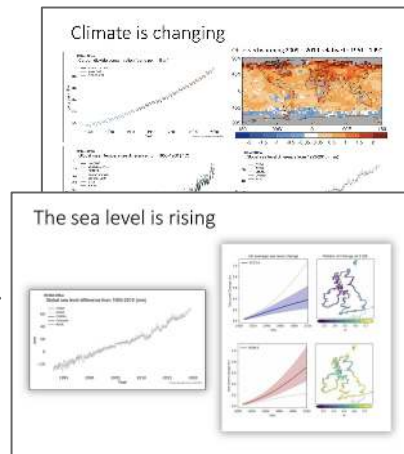
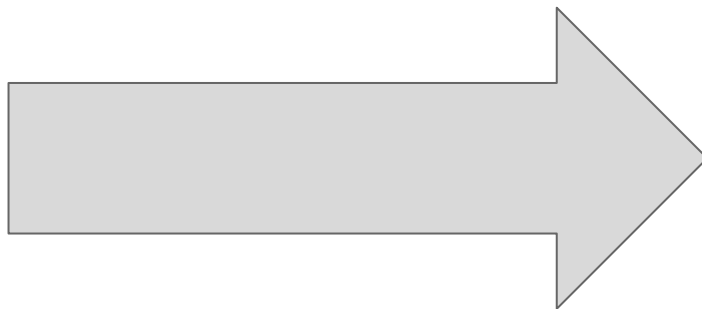
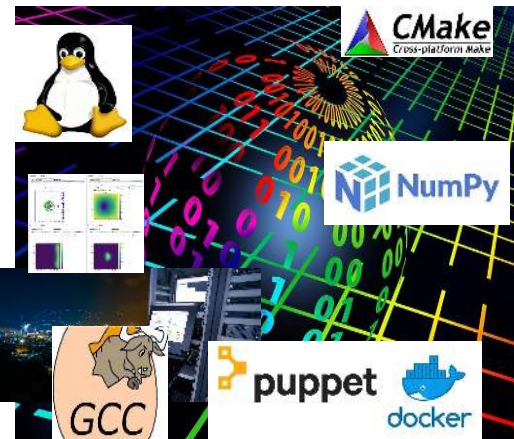
# Reproducibility and Replicability

**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.

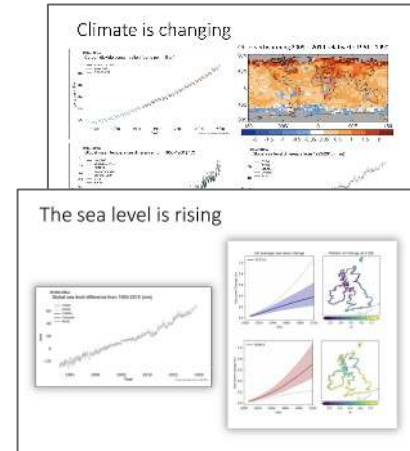
“Unreproducible Research is Reproducible”, Bouthillier et al., ICML 2019



# 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>

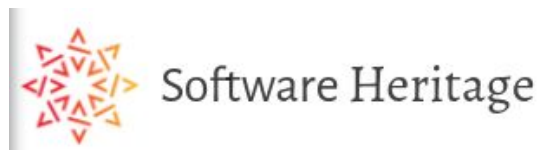
Reproducible Science is good. Replicated Science is better.

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](#)

# ICPE 2022

13th ACM/SPEC International Conference on Performance Engineering

## 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 Research Artifacts track). The PDF should contain a stable URL (or DOI) to the artifacts. The URL 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.

# Why should you make your software (work) reproducible and replicable?

Why not: hard to trust your results; hard to innovate on top of your work; impossible to refute hypothesis or verify new theories

Easier to redo computation when eg inputs change

Easier to inspect and document

Easier to evolve and maintain

Openness and transparency boost innovation

It can be seen as **engineering high-quality software**

It is an important effort, it's very hard! But worth, not to say mandatory...



What's your experience with reproducible science so far?

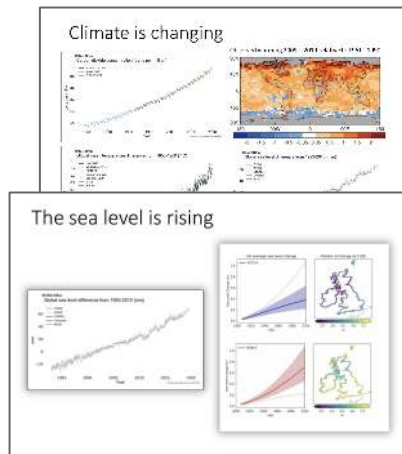
# Reproducible science

*“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

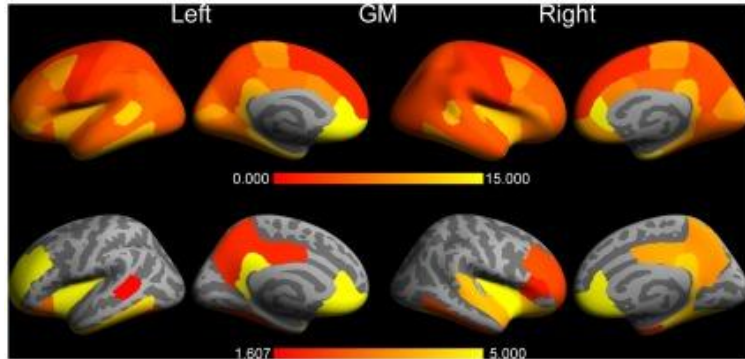


# Should software version numbers determine science?

> [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 <sup>1</sup>, Petra Habets, Heidi I L Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, Machteld Marcelis

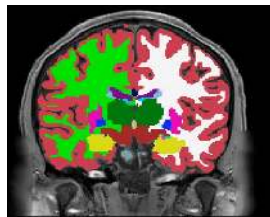


Significant differences were revealed between FreeSurfer version v5.0.0 and the two earlier versions. [...] About a factor two smaller differences were detected between Macintosh and Hewlett-Packard workstations and between OSX 10.5 and OSX 10.6. The observed differences are similar in magnitude as effect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



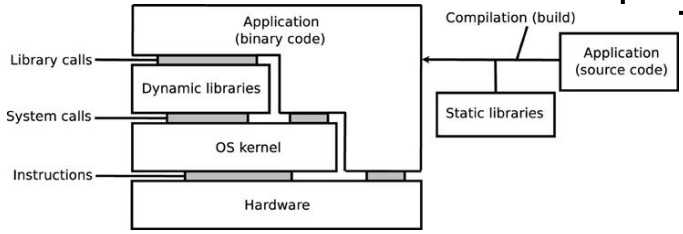
The implementation of mathematical functions manipulating single-precision **floating-point numbers in libmath** has evolved during the last years, leading to numerical differences in computational results. While these differences have little or no impact on simple analysis pipelines such as brain extraction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

	<b>Cluster A</b>	<b>Cluster B</b>
Applications	Freesurfer 5.3.0, build <b>1</b> FSL 5.0.6, build <b>1</b> CIVET 1.1.12-UCSF, build <b>1</b>	Freesurfer 5.3.0, build <b>1</b> and <b>2</b> FSL 5.0.6, build <b>1</b> and <b>2</b> CIVET 1.1.12-UCSF, build <b>1</b>
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

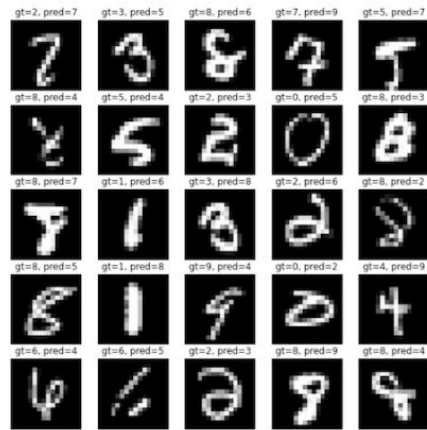
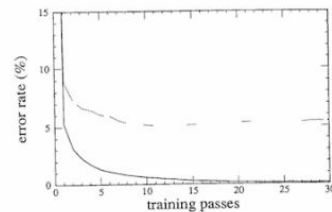
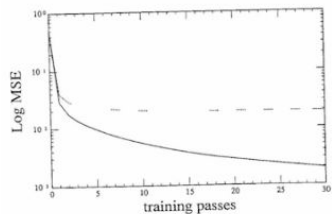
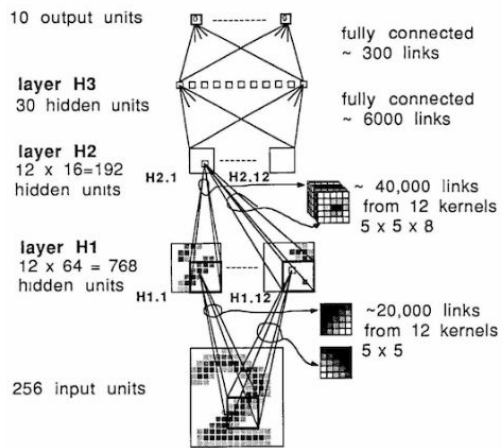
Statically building programs improves reproducibility across OSes, but small differences may still remain when dynamic libraries are loaded by static executables[...]. When static builds are not an option, software heterogeneity might be addressed using virtual machines. However, such solutions are only 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)

# Is it possible to reproduce a paper from 1987?

## lecun1989-repro



This code tries to reproduce the 1989 Yann LeCun et al. paper: [Backpropagation Applied to Handwritten Zip Code Recognition](#). To my knowledge this is the earliest real-world application of a neural net trained with backpropagation (now 33 years ago).

<https://github.com/karpathy/lecun1989-repro>

# Is it possible to reproduce a paper from 1987?

**input data**



run

Since we don't have the exact dataset that was used in the paper, we take MNIST and randomly pick examples from it to generate an approximation of the dataset, which contains only 7291 training and 2007 testing digits, only of size 16x16 pixels (standard MNIST is 28x28).

<https://github.com/karpathy/lecun1989-repro>

# Is it possible to reproduce a paper from 1987?

## same results? well...

Now we can attempt to reproduce the paper. The original network trained for 3 days, but my (Apple Silicon M1) MacBook Air 33 years later chunks through it in about 90 seconds. (non-emulated arm64 but CPU only, I don't believe PyTorch and Apple M1 are best friends ever just yet, but anyway still about 3000X speedup). So now that we've run prepro we can run repro! (haha):

```
$ python repro.py
```

Running this prints (on the 23rd, final pass):

```
eval: split train. loss 4.073383e-03. error 0.62%. misses: 45  
eval: split test . loss 2.838382e-02. error 4.09%. misses: 82
```

This is close but not quite the same as what the paper reports. To match the paper exactly we'd expect the following instead:

```
eval: split train. loss 2.5e-3. error 0.14%. misses: 10  
eval: split test . loss 1.8e-2. error 5.00%. misses: 102
```

<https://github.com/karpathy/lecun1989-repro>



# Is it possible to reproduce a paper from 1987?

huge variants/decisions space!

hyperparameter?

algorithm?

implementation?

NN architecture?

Open questions:

- The 12 -> 8 connections from H2 to H1 are not described in this paper... I will assume a sensible block structure connectivity
- Not clear what exactly is the "MSE loss". Was the scaling factor of 1/2 included to simplify the gradient calculation? Will assume no.
- What is the learning rate? I will run a sweep to determine the best one manually.
- Was any learning rate decay used? Not mentioned, I am assuming no.
- Was any weight decay used? not mentioned, assuming no.
- Is there a bug in the pdf where in weight init the fan in should have a square root? The pdf's formatting is a bit messed up. Assuming yes.
- The paper does not say, but what exactly are the targets? Assuming they are +1/-1 for pos/neg, as the output units have tanh too...

One more notes on the weight init conundrum. Eg the "Kaiming init" is:

```
a = gain * sqrt(3 / fan_in)
-U(-a, a)
```

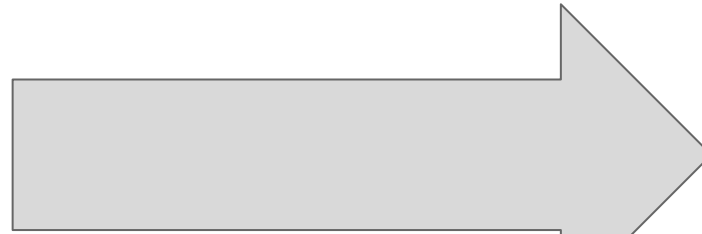
For tanh neurons the recommended gain is 5/3. So therefore we would have  $a = \sqrt{3} * 5 / 3 * \sqrt{1 / fan\_in} = 2.89 * \sqrt{1 / fan\_in}$ , which is close to what the paper does (gain 2.4). So if the original work in fact did use a sqrt and the pdf is just formatted wrong, then the (modern) Kaiming init and the originally used init are pretty close.

<https://github.com/karpathy/lecun1989-repro>

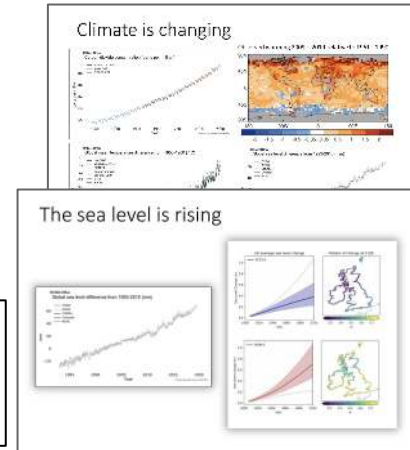
# Reproducible science as a (deep) software variability problem

*“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





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!

# General advice: #1 Open source software and tools

Open format over tools

~~Excel? Word? Evernote?~~

eg CSV+Pandas, Markdown, orgmode, Joplin

~~SAS, Matlab, Mathematica~~

eg Python, R, scilab

~~Dropbox, Google drive~~

eg git/gitlab, Zenodo, SoftwareHeritage

## **General advice: #2 Reproducibility mindset**

Try to design your project and research in such a way it will be reproducible  
reusable, modular, testable, parameterizable software

Try to document your effort and trace your decisions (eg cahier de laboratoires,  
versioning systems)

Don't wait the last minute to prepare a shareable repository (share it with your  
co-authors as soon as possible!)

From notebooks (more exploratory) to engineered, scalable software

# Notebook

jupyter ML\_training\_and\_reproducibility Last Checkpoint: 06/26/2023 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [1]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

In [2]: # data to be used -> UCI heart disease
# from the university of California
# data downloaded via https://github.com/sagnikghoshcr7/Heart-Disease-Prediction
# seems ok with the description from UCI -> https://archive.ics.uci.edu/dataset/45/heart+disease

In [3]: dataset = pd.read_csv('heart.csv')
X = dataset.drop('target', axis=1)
y = dataset['target'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1)

In [4]: # look if split looks okayish
print(X)
print(y)
print(X_train)
print(X_test)

count = np.sum(y_test == 1)
print("total number of instance in test: " + str(len(y_test)))
print(y_test)
print("total number of instance in test with class 1: "+str(count))

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0    63   1   3     145    233   1         0     150     0     2.3
1    37   1   2     130    250   0         1     187     0     3.5
2    41   0   1     130    204   0         0     172     0     1.4
3    56   1   1     120    236   0         1     178     0     0.8
4    57   0   0     120    354   0         1     163     1     0.6
...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
298  57   0   0     140    241   0         1     123     1     0.2
299  45   1   3     110    264   0         1     132     0     1.2
300  68   1   0     144    193   1         1     141     0     3.4
301  57   1   0     130    131   0         1     115     1     1.2
302  57   0   1     130    236   0         0     174     0     0.0

      slope  ca  thal
0         0   0   1
1         0   0   2
2         2   0   2
3         2   0   2
4         2   0   2
...  ...  ...  ...
298        1   0   3
299        1   0   3
300        1   2   3
301        1   1   3
302        1   1   2
```



# Git/Gitlab|Github

## Languages

- Jupyter Notebook 99.7%
- Shell 0.2%
- Python 0.1%
- TeX 0.0%
- Dockerfile 0.0%
- Makefile 0.0%



master

input\_sensitivity / results / RQS / RQ3 / RQ3.ipynb

File explorer showing repository structure:

- data: add imagemagick results (2 years ago)
- replication: up container (7 months ago)
- results: Open Science Initiative JSS 3 - add RQS markdowns (5 months ago)
- src: up RQ2 JSS (9 months ago)
- JSS\_Input\_Sensitivity.pdf: up Open Science Initiative JSS 1 (5 months ago)
- LICENSE: Create LICENSE (6 months ago)
- README.md: Update README.md (5 months ago)

README.md content:

### Input Sensitivity

This is the companion repository related to our paper "Input Sensitivity on the Performance of Configurable Systems: An Empirical Study" submitted to the Journal of Software and Systems.

### Organization

Measurements and details about data can be consulted in the **data** folder.

Source code can be found in the **src** directory.

The **results** folder contains the results shown in the submission, as well as complementary results.

The **replication** folder contains all you need to replicate our experiments.

The article (pdf file) can be consulted directly in the root directory.

## research question results

2 study how inputs affect (1) performance distributions could change in a negligible way, with input sensitivity, it is necessary to quantify how replicating the software to its input data is worth the cost

## Can we ignore input sensitivity

How much we can lose, we first define two scenarios. In this scenario, we just train a simple performance configuration according to the model, configure the relative

- S2 - Ignoring input sensitivity. In this scenario, we train a model on a given input i.e. the source input, and then predict the best configuration for this source input. If we ignore the threat of input sensitivity, we can easily reuse this model for any other input, including the target input defined in S1. Finally, we execute the software with the configuration predicted by our model on the target input

```

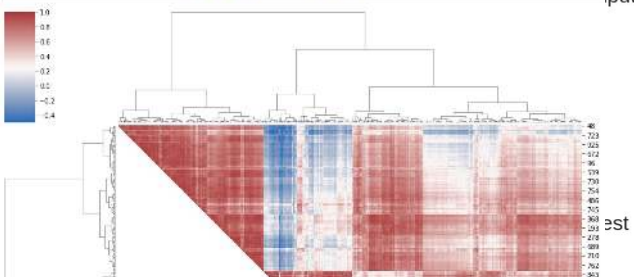
A1 = True
    node[order[1]][order[2]] = True

sns.relplot(x="dir", y="perf", hue="method",
            ci="sd",
            plot_df=plot_df,
            style="method",
            facet_kinds={("dir", "perf")})

return out_dir, perf

method = 'wand'
groups = plot_correlation_matrix(demo_data, corr_data,
                               'corr_data_ign-deroo-Spearman' + predDimension + '.pdf',
                               method)
for k in range(10, 10+1):

```



of, the  
on the real  
ask  
input

# **General advice: #3 Master your computing environment**

Don't assume "it works on my machine"

Track dependencies, versions, packages, tools

Ask yourself: "What if I have to reproduce my results?"

Ask your co-authors to reproduce your results (ideally throughout the work, not at the end)

Automate the synthesis of your computing environment

Test your computing environment!



# Computing environment matters!

binaries

configure; make

reprozip

Docker

Guix

Nix

What's the problem of only sharing binaries and executable?

~~What's the problem of only sharing binaries and executable?~~

What's the problem of...

`./configure; make; make install`

# What's the problem with notebooks?

```
jupyter ML_training_and_reproducibility Last Checkpoint: 06/26/2023 (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3.0
In [1]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

In [2]: # data to be used -> UCI heart disease
# from the university of California
# data downloaded via https://github.com/sagnikghoshcr7/Heart-Disease-Prediction
# seems ok with the description from UCI -> https://archive.ics.uci.edu/dataset/45/heart+disease

In [3]: dataset = pd.read_csv('heart.csv')
X = dataset.drop('target', axis=1)
y = dataset['target'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1)

In [4]: # look if split looks okayish
print(X)
print(y)
print(X_train)
print(X_test)

count = np.sum(y_test == 1)
print("total number of instance in test: " + str(len(y_test)))
print(y_test)
print("total number of instance in test with class 1: "+str(count))

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0    63   1   3     145    233   1         0     150     0     2.3
1    37   1   2     130    250   0         1     187     0     3.5
2    41   0   1     130    204   0         0     172     0     1.4
3    56   1   1     120    236   0         1     178     0     0.8
4    57   0   0     120    354   0         1     163     1     0.6
...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
298  57   0   0     140    241   0         1     123     1     0.2
299  45   1   3     110    264   0         1     132     0     1.2
300  68   1   0     144    193   1         1     141     0     3.4
301  57   1   0     130    131   0         1     115     1     1.2
302  57   0   1     130    236   0         0     174     0     0.0

      slope  ca  thal
0         0   0   1
1         0   0   2
2         2   0   2
3         2   0   2
4         2   0   2
...  ...  ...  ...
298        1   0   3
299        1   0   3
300        1   2   3
301        1   1   3
302        1   1   2
```

# Requirements (wishlist!)

Source code under version control tools like git.

Versions of all its dependencies (ideally the source code of dependencies)

Compiler/interpreter and on the options it has been passed

Environment variables

Parameters of your application

Ecosystem agnostic

...





# Full computational reproducibility!

ReproZip can automatically pack your research along with all necessary data files, libraries, environment variables and options into a self-contained bundle. Then ReproZip can use that bundle to automatically set up the same original environment so anybody can reproduce the research on a different machine, without tracking down and installing the dependencies, or even having to run the same operating system!

[LEARN MORE »](#)

## No lock-in

ReproZip works locally on your computer, server, or high performance computing clusters. Once you make an `.rpz` bundle, you can reproduce it with a variety of unpacker plug-ins (e.g. Docker, Vagrant, Singularity) or in the cloud with [ReproServer](#).

[TRY OUT REPROZIP TODAY »](#)

## Free & Open Source

The development of ReproZip has always been open source and free to use since the project began in 2013. We welcome and encourage anyone to submit pull requests or issues with any feature requests or bug reports!

[CHECK OUT THE GITHUB REPO »](#)

## Extensible

ReproZip has been used to capture a wide variety of digital materials, from desktop applications and scripts in various languages to dynamic websites to high performance computing experiments. You can then reproduce your work across various OS's!

[EXPLORE SOME EXAMPLES »](#)



master ▾

[input\\_sensitivity](#) / [replication](#) / [containers](#) / [nodejs](#) / Dockerfile 

llesoil replication containers

Code

Blame

20 lines (18 loc) · 669 Bytes

```
1 FROM fedora:33
2 # update packages
3 RUN dnf -y up --exclude=kernel --exclude=kernel-core --exclude=kernel-devel --exclude=kernel-headers --exclude=python --exclude=gcc-c++ --exclude=make
4 # https://github.com/nodejs/node/blob/master/BUILDING.md
5 RUN dnf install -y python gcc-c++ make wget nano
6 RUN uname -r >> infos \
7     && gcc --version >> infos \
8     && make -v >> infos \
9     && python --version >> infos
10 # clone nodejs
11 RUN wget https://github.com/nodejs/node/archive/refs/tags/v15.14.0.tar.gz \
12     && tar -xf v15.14.0.tar.gz
13 # command to compile node with the default configurations
14 RUN cd node-15.14.0 \
15     && ./configure \
16     && make -j4
17 # Add files to run the scripts
18 COPY . /
--
```

<https://nixos.org/>

---

```
with import <nixpkgs> {}; stdenv.mkDerivation {
  name = "hello";

  # Source Code
  # See: https://nixos.org/nixpkgs/manual/#ssec-unpack-phase
  src = ./src;

  # Dependencies
  # See: https://nixos.org/nixpkgs/manual/#ssec-stdenv-dependencies
  buildInputs = [ coreutils gcc ];

  # Build Phases
  # See: https://nixos.org/nixpkgs/manual/#sec-stdenv-phases
  configurePhase = ''
    declare -xp
  '';
  buildPhase = ''
    gcc "$src/hello.c" -o ./hello
  '';
  installPhase = ''
    mkdir -p "$out/bin"
    cp ./hello "$out/bin/"
  '';
}
```



<https://guix.gnu.org/>

```
(define-public hello
  (package
    (name "hello")
    (version "2.10")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-" version
                                   ".tar.gz"))
              (sha256
                (base32
                 "0ssi1wpaf7plawqqjwigppsg5fyh99vdlb9kzl7c9lmg89ndq1i")))))
    (build-system gnu-build-system)
    (synopsis "Hello, GNU world: An example GNU package")
    (description
     "GNU Hello prints the message \"Hello, world!\" and then exits. It
     serves as an example of standard GNU coding practices. As such, it supports
     command-line arguments, multiple languages, and so on.")
    (home-page "https://www.gnu.org/software/hello/")
    (license gpl3+)))
```

# General advice: #4 Test and doubt about the quality of your software

Don't assume "it works"

Test here means **automated test**: manually checking that a function is doing what you expect is OK for prototyping/exploring, but it's limited in terms of input, number of runs, and well it's hard to redo the effort for any evolution

small-scale: "unit" test of a function; test a script (eg that preprocesses data);  
interaction between "modules" of your software

large-scale: test the whole processing chains

Test even the libraries and tools you depend on!

# General advice: #5 Automate everything

Automate the retrieval of data

Automate the pre-process of data

Automate the build of software

Automate the running/measurement (incl. deployment) of software

Automate the synthesis of the computing environment

Automate the production of figures (numbers)

Automate the production of Figures/Tables/...

Automate your automation: generate programs that generate what you want

# General advice: #6 Threats to validity

Report on threats to validity (see deep variability)

Describe how you mitigate the threats

Discuss threats!

Internally: think about how your threats relate to your software or data

# General advice: #7 Explore the variants space

Make vary your experiments

Robustness of your results

Opportunity to find new paths/hypothesis/...

Sometimes called ablation studies or “sensitivity” studies or ...

It's much easier with a reproducible software stack ;)

# General advice: #4 Test and doubt about the quality of your software

Don't assume "it works"

Test here means **automated test**: manually checking that a function is doing what you expect is OK for prototyping/exploring, but it's limited in terms of input, number of runs, and well it's hard to redo the effort for any evolution

small-scale: "unit" test of a function; test a script (eg that preprocesses data);  
interaction between "modules" of your software

large-scale: test the whole processing chains

Test even the libraries and tools you depend on!

# Testing Problem in a nutshell

#1 Test Generation Problem (input)

#2 Oracle Problem (output)

Input

System under  
Study

Output



# Testing techniques for reproducible science

#1 Test Generation Problem (input)

#2 Oracle Problem (output)

Important to ensure high-quality software... Test your script, your module, your component, your workflow, your generator... Small-scale and large-scale!

Input

System under  
Study

Output

# 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)

Let say we inject fault on purpose (~manual mutant)

**Listing 1.** A Python implementation of bubble\_sort affected by a bug when swapping array elements.

```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i]🐛
    return arr
```

# Assertion-based Testing

**Listing 1.** A Python implementation of `bubble_sort` affected by a bug when swapping array elements.

```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i] 🐛
    return arr
```

**Listing 2.** A manually-written unit test.

```
arr = [3, 1, 2]
bubble_sort(arr)
assert arr == [1, 2, 3] # AssertionError, ←
    actual: [1, 2, 1]
```

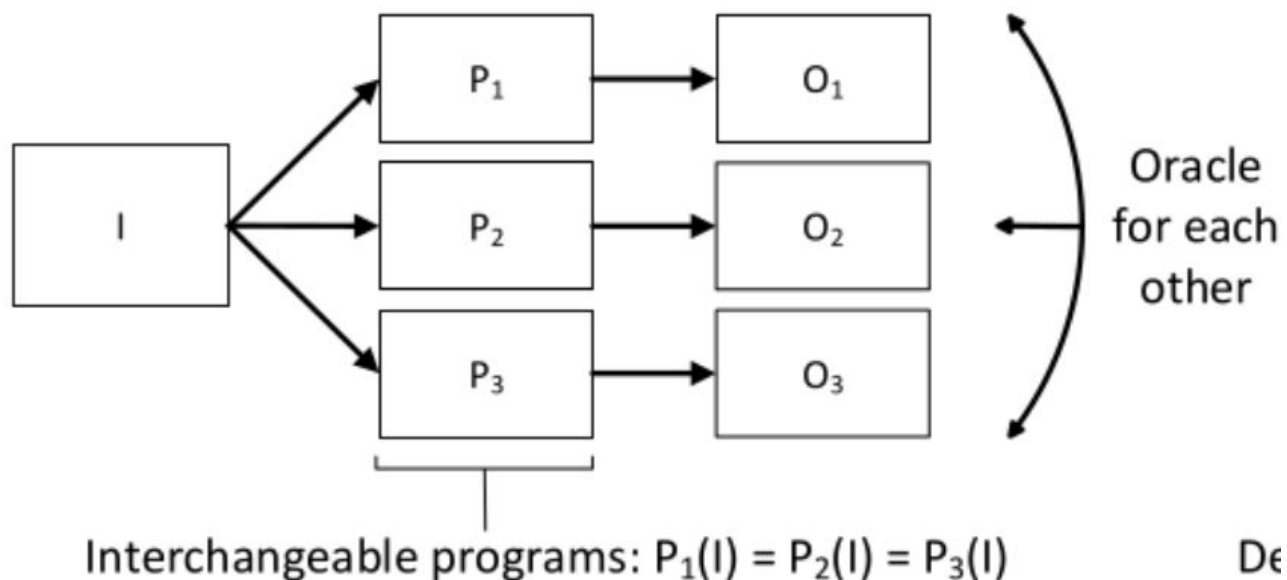
# Strengths and Limits of Assertion-based Testing

- Precise oracles
- Domain-specific knowledge
- Examples-based
  - Input test is limited by construction
  - Incomplete (cases)
- Tests subject to developers bias
- Costly
  - One specific oracle per example (labelling)


## **Listing 2.** A manually-written unit test.

```
arr = [3, 1, 2]
bubble_sort(arr)
assert arr == [1, 2, 3] # AssertionError, ←
    actual: [1, 2, 1]
```

## Differential Testing



**Listing 1.** A Python implementation of `bubble_sort` affected by a bug when swapping array elements.

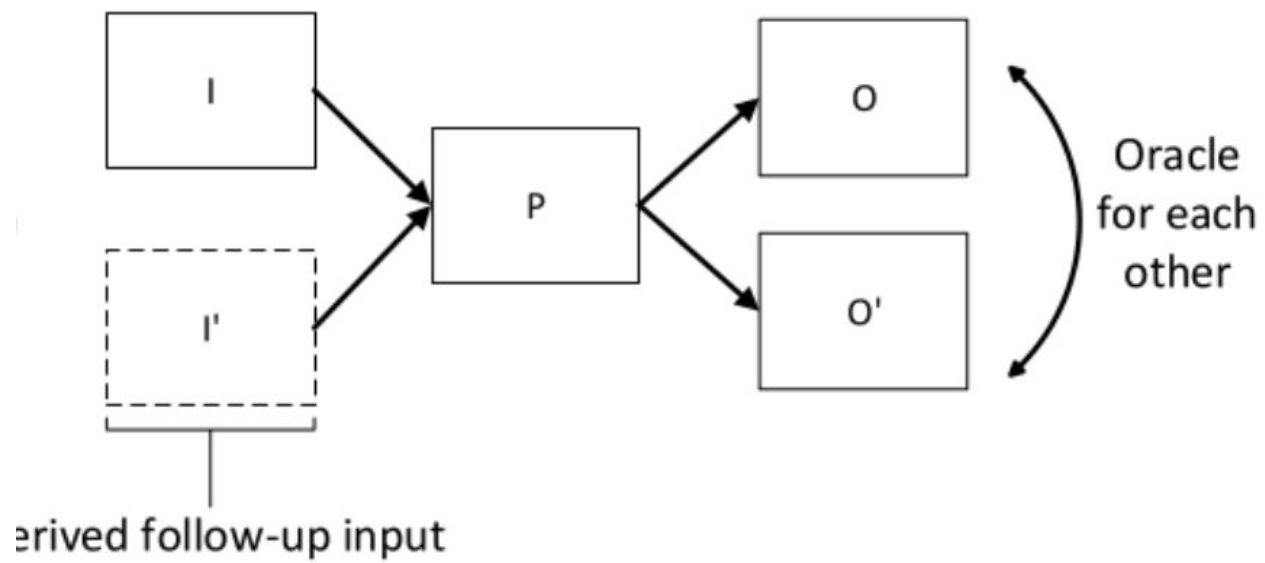
```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i] 
    return arr
```

**Listing 3.** Differential testing using multiple implementations of sorting algorithms.


```
sorting_algorithms = [bubble_sort, merge_sort, ↵
                      insertion_sort]
while True:
    arr = get_random_array() # e.g., [3, 1, 2]
    sorted_arrays = [alg(arr.copy()) for alg in ↵
                    sorting_algorithms]
    all_same = all(sorted_arr == sorted_arrays[0] ↵
                  for sorted_arr in sorted_arrays)
    assert all_same
```



## Metamorphic Testing




**Listing 1.** A Python implementation of `bubble_sort` affected by a bug when swapping array elements.

```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i] 
    return arr
```

**Listing 4.** Metamorphic testing by comparing whether the relative order is maintained for a smaller array.

```
while True:
    arr = get_random_array()
    if len(arr) >= 1:
        sorted_arr = bubble_sort(arr.copy())
        random_elem = random.choice(sorted_arr)
        arr.remove(random_elem)
        sorted_smaller_arr = bubble_sort(arr)
        sorted_arr.remove(random_elem)
        assert sorted_arr == sorted_smaller_arr
```

**Listing 1.** A Python implementation of `bubble_sort` affected by a bug when swapping array elements.

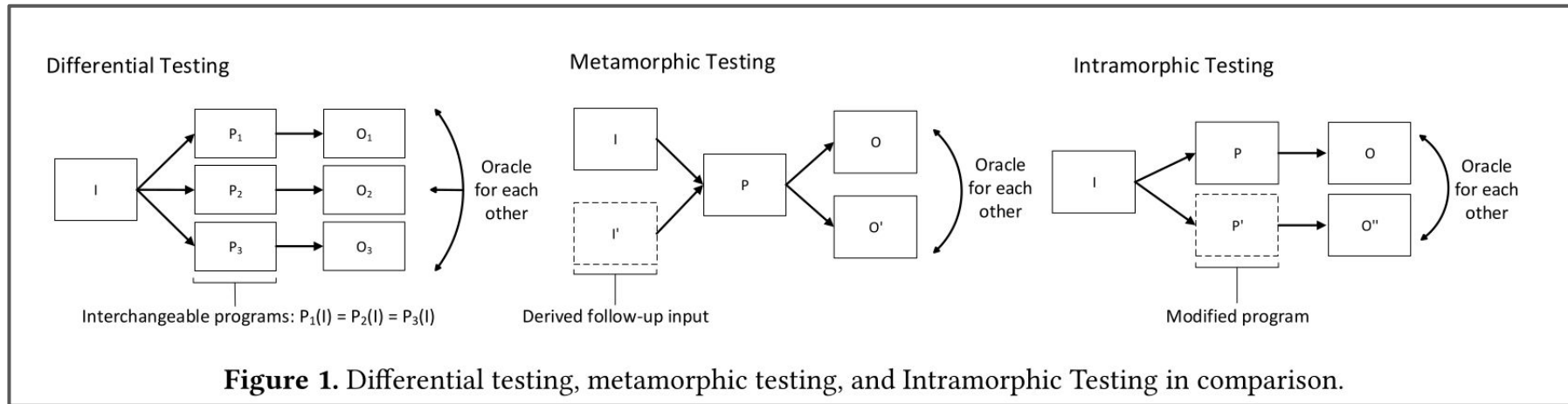
```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i] 
    return arr
```

As Listing 4 shows, we designed a MR that checks whether the relative order of sorted elements is maintained when an element is removed from an input array. For example, given an input array  $i1 = [3, 1, 2]$  and a correctly-sorted array  $o1 = [1, 2, 3]$ , we derive a new input by removing one element  $e$  from the input array, for example,  $e = 2$ , resulting in a new input array  $i2 = [3, 1]$ , for which we can infer the expected result by removing  $e$  from  $o1$ , that is,  $o2 = [1, 3]$ . This specific idea enables finding the bug as well. When passing  $[3, 1, 2]$  as input array, the incorrect output  $[1, 2, 1]$  is produced; when passing  $[3, 1]$  as input by removing 2, the output is  $[1, 3]$ , rather than  $[1, 1]$ , breaking

**Listing 4.** Metamorphic testing by comparing whether the relative order is maintained for a smaller array.

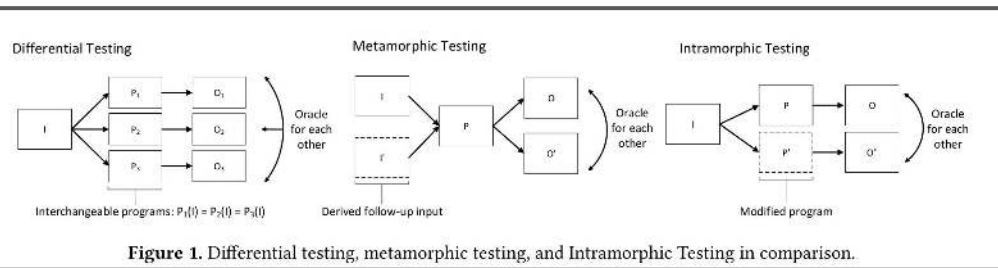
```
while True:
    arr = get_random_array()
    if len(arr) >= 1:
        sorted_arr = bubble_sort(arr.copy())
        random_elem = random.choice(sorted_arr)
        arr.remove(random_elem)
        sorted_smaller_arr = bubble_sort(arr)
        sorted_arr.remove(random_elem)
        assert sorted_arr == sorted_smaller_arr
```

**Intramorphic Testing.** In this work, we propose Intramorphic Testing to tackle the test oracle problem by changing the system under test so that, for a given input and the original system's output, an oracle for the output of the changed system can be derived. Figure 1 illustrates the approach. Given a program  $P$ , a new program  $P'$  is derived for which, given an input  $I$ , a known relationship between the two program's outputs (*i.e.*,  $O = P(I)$  and  $O' = P'(I)$ ) is validated. Similar to metamorphic testing, Intramorphic Testing is a high-level idea and conceptualization, for which many instantiations are possible.



**Listing 1.** A Python implementation of bubble\_sort affected by a bug when swapping array elements.

```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i]
    return arr
```

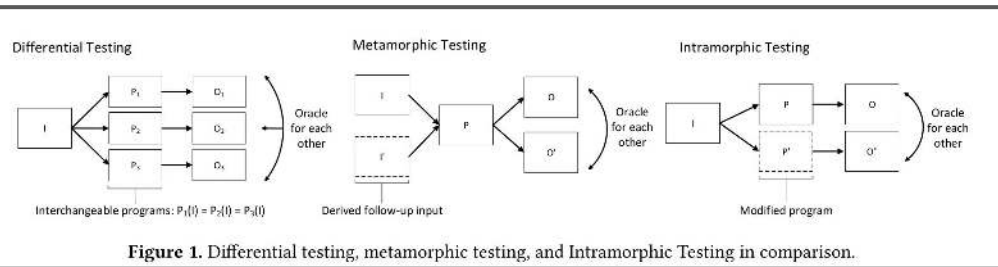


**Listing 5.** An Intramorphic Testing realization that adds an additional reverse sorting function, whose reversed output array is compared with the sorted output array of the original sorting function.

```
while True:
    arr = get_random_array()
    sorted_arr = bubble_sort(arr.copy())
    reverse_sorted_arr = bubble_sort_reverse(arr.↔
        copy())
    sorted_arr.reverse()
    assert sorted_arr.reverse() == ↔
        reverse_sorted_arr
```

**Listing 1.** A Python implementation of `bubble_sort` affected by a bug when swapping array elements.

```
def bubble_sort(arr):
    length = len(arr)
    for i in range(length):
        for j in range(0, length - i - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[i]
    return arr
```



**Figure 1.** Differential testing, metamorphic testing, and Intramorphic Testing in comparison.

**Listing 5.** An Intramorphic Testing realization that adds an additional reverse sorting function, whose reversed output array is compared with the sorted output array of the original sorting function.

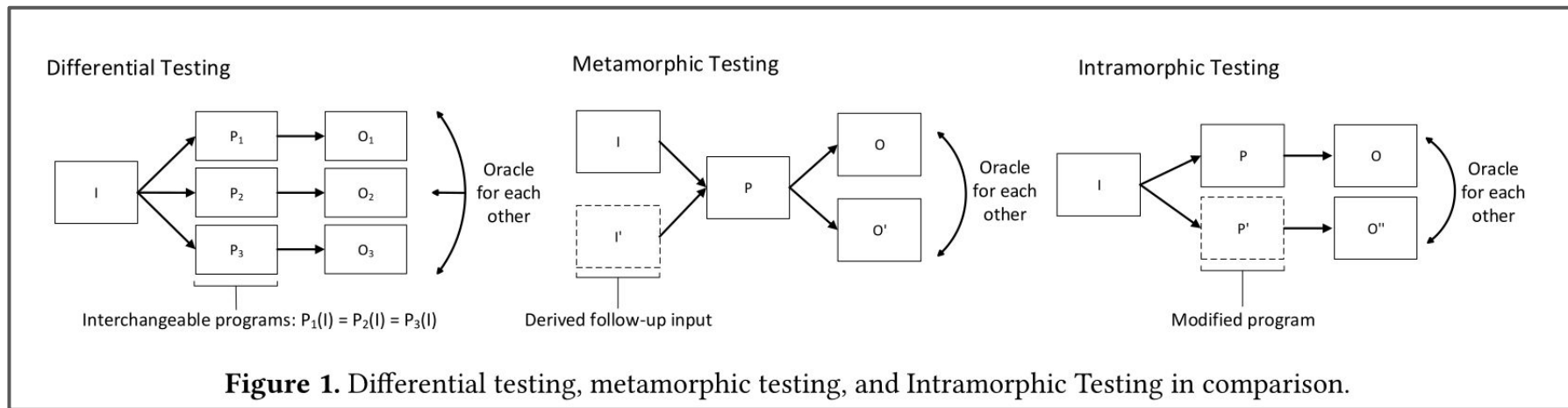
```
while True:
    arr = get_random_array()
    sorted_arr = bubble_sort(arr.copy())
    reverse_sorted_arr = bubble_sort_reverse(arr.↔
        copy())
    sorted_arr.reverse()
    assert sorted_arr.reverse() == ↔
        reverse_sorted_arr
```

To realize an Intramorphic Testing technique for our use case, we could implement another alternative sorting implementation `bubble_sort_reverse()` as a potential replacement for `bubble_sort()` that sorts the array in descending order. Thus, the expectation that we could check is that, by reversing one of the two output arrays, two equivalent arrays are obtained (see Listing 5). For example, for an input `[3, 1, 2]`, we would expect an output `[3, 2, 1]` for `bubble_sort_reverse()` rather than `[1, 2, 3]`. This concrete Intramorphic Testing realization also detects the bug. Even if `bubble_sort_reverse()` is affected by the same index bug (i.e., by replacing only the comparison operator in Listing 1), it would detect the bug, since `bubble_sort()` returns `[3, 2, 3]` as

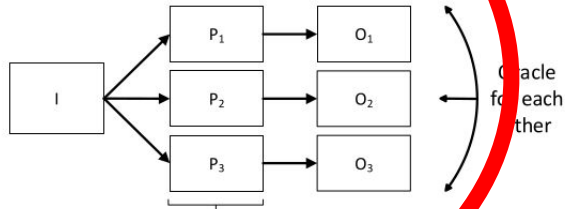


## Abstract

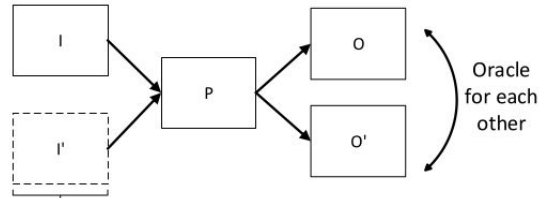
A *test oracle* determines whether a system behaves correctly for a given input. Automatic testing techniques rely on an automated test oracle to test the system without user interaction. Important families of automated test oracles include *Differential Testing* and *Metamorphic Testing*, which are both black-box approaches; that is, they provide a test oracle that is oblivious to the system's internals. In this work, we propose *Intramorphic Testing* as a white-box methodology to tackle the test oracle problem. To realize an Intramorphic Testing approach, a modified version of the system is created, for which, given a single input, a test oracle can be provided that relates the output of the original and modified systems. As a concrete example, by replacing a greater-equals operator in the implementation of a sorting algorithm with smaller-equals, it would be expected that the output of the modified implementation is the reverse output of the original implementation. In this paper, we introduce the methodology and illustrate it via a set of use cases.



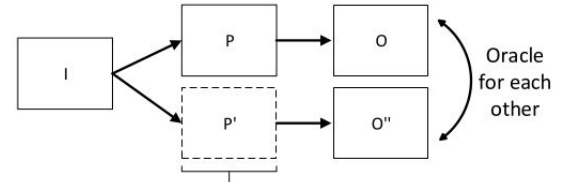
### Differential Testing



### Metamorphic Testing



### Intramorphic Testing



**Figure 1.** Differential testing, metamorphic testing, and Intramorphic Testing in comparison.

**Random Generator:  
Csmith**



*C program*



**gcc -O0**



**gcc -O2**



**clang -Os**

...

*results*

**vote**

*majority*

*minority*



# Finding and Understanding Bugs in C Compilers

Xuejun Yang   Yang Chen   Eric Eide   John Regehr

University of Utah, School of Computing

{jxyang, chenyang, eeide, regehr}@cs.utah.edu

- *Randomized* differential testing has the advantage that no oracle for test results is needed.
- Idea: if one has multiple, deterministic implementations of the same specification, all implementations must produce the same result from the same valid input
- 300+ bugs found in compilers (eg gcc)

```
1 int foo (void) {  
2     signed char x = 1;  
3     unsigned char y = 255;  
4     return x > y;  
5 }
```

**Figure 1.** We found a bug in the version of GCC that shipped with Ubuntu Linux 8.04.1 for x86. At all optimization levels it compiles this function to return 1; the correct result is 0. The Ubuntu compiler was heavily patched; the base version of GCC did not have this bug.

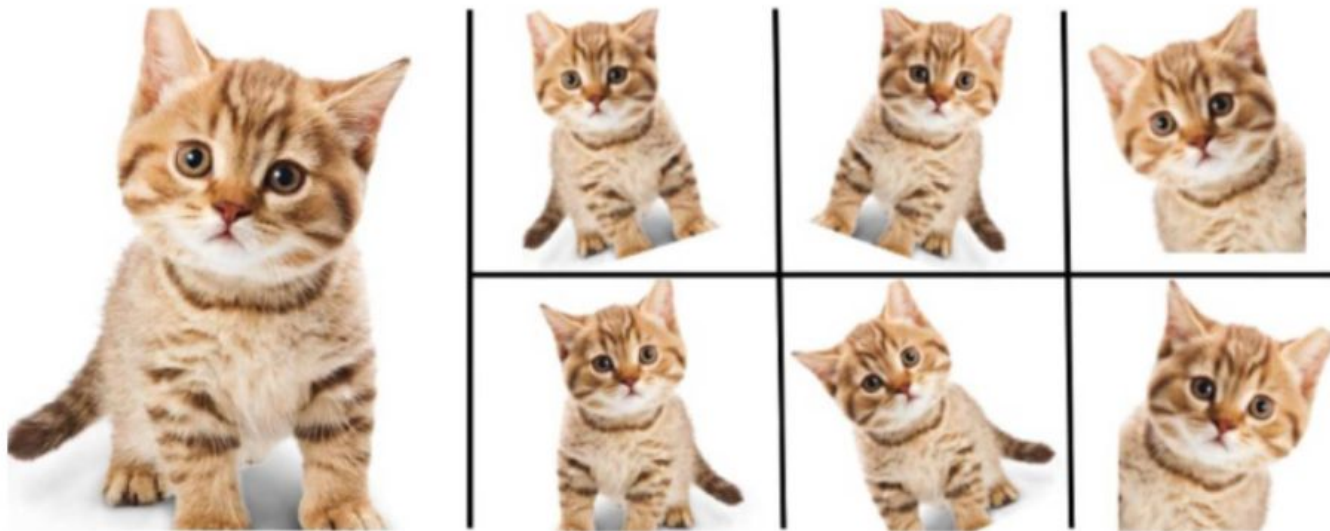
Given an image, you can:

- change its colour palette
- flip it
- rotate it
- zoom...

And still know what the output should be!

**Free test-cases (input)**

**Don't need to specify the oracle (output)**



# Metamorphic Testing for CV

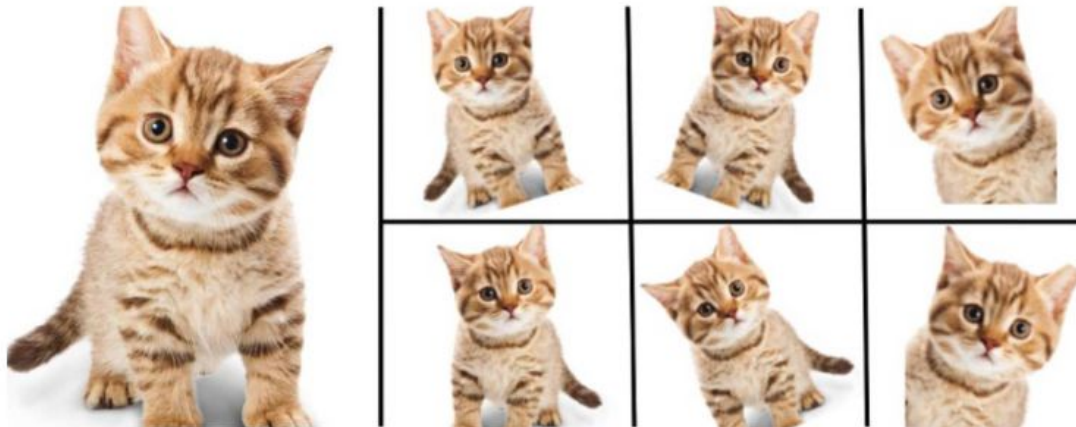
Given an image, you can:

- change its colour palette
- flip it
- rotate it
- zoom...

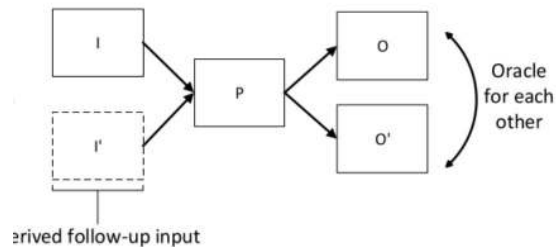
And still know what the output should be!

**Free test-cases (input)**

**Don't need to specify the oracle (output)**



Metamorphic Testing



# Testing toolbox

- Testing Problem
  - Test Generation Problem (input)
  - Oracle Problem (output)
- Assertion-based testing
- Mutation Testing
- Differential Testing
- Metamorphic Testing
- Multimorphic Testing
- Many testing topics (property-based testing, search-based testing, fuzzing, etc.)
  - see references!

# References

- Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, Shin Yoo: The Oracle Problem in Software Testing: A Survey. *IEEE Trans. Software Eng.* 41(5): 507-525 (2015)
- Oscar Luis Vera-Pérez, Benjamin Danglot, Martin Monperrus, Benoit Baudry: A comprehensive study of pseudo-tested methods. *Empirical Software Engineering* 24(3): 1195-1225 (2019)
- Xuejun Yang, Yang Chen, Eric Eide, John Regehr: Finding and understanding bugs in C compilers. *PLDI 2011*: 283-294
- Vu Le, Mehrdad Afshari, Zhendong Su: Compiler validation via equivalence modulo inputs. *PLDI 2014*: 216-226
- Paul Temple, Mathieu Acher, and Jean-Marc Jézéquel. *Empirical Assessment of Multimorphic Testing (2019)*. In *IEEE Transactions on Software Engineering (TSE)*



# References

- Alastair F. Donaldson, Hugues Evrard, Andrei Lascu, Paul Thomson: Automated testing of graphics shader compilers. PACMPL 1(OOPSLA): 93:1-93:29 (2017)
- José A. Galindo, Mauricio Alferez, Mathieu Acher, Benoit Baudry, and David Benavides. A Variability-based Testing Approach for Synthesizing Video Sequences (2014). In International Symposium on Software Testing and Analysis (ISSTA'14)
- Sergio Segura, José Antonio Parejo, Javier Troya, Antonio Ruiz Cortés: Metamorphic Testing of RESTful Web APIs. IEEE Trans. Software Eng. 44(11): 1083-1099 (2018)
- Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack (2018). In Empirical Software Engineering

# General advice: #7 Explore the variants space

Make vary your experiments

Robustness of your results

Opportunity to find new paths/hypothesis/...

Sometimes called ablation studies or “sensitivity” studies or ...

It's much easier with a reproducible software stack ;)

# AGENDA

Reproducible Science and (Deep) Software (Variability)

**Deep Software Variability**

Evidence of Deep Software Variability in Science

Threats and Opportunities

# 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)

# deep software variability



hardware variability

15,000+ options

thousands of compiler  
flags and compile-time

options  
dozens of  
preferences

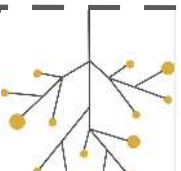
100+ command-line  
parameters

1000+ feature toggles

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  
(replicable)

Variability  
Output

(scientific results  
most of the time  
quantitative  
information)

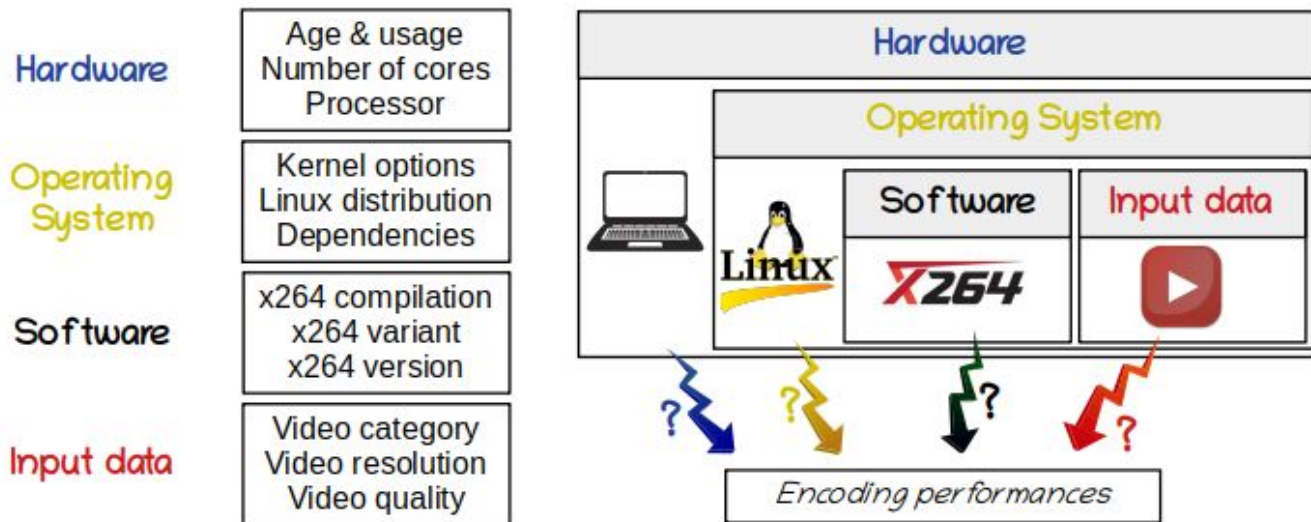


264

# Deep software variability: does it matter? *i.e.* Are layers/features orthogonal or are there interactions?

*Variability layers*

**264** 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]  
Overridden by user settings.  
- ultrafast,superfast,veryfast,faster,fast  
- medium,slow,slower,veryslow,placebo

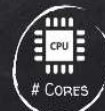
--tune <string> Tune the settings for a particular type of source or situation  
Overridden by user settings.  
Multiple tunings are separated by commas.  
Only one psy tuning can be used at a time.

-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)

**HARDWARE**

**OPERATING SYSTEM**

**SOFTWARE**

**INPUT DATA**



(A) DELL LATITUDE 7400



(B) RASPBERRY PI 4 MODEL B



20.04



10.4

x264 (1) --mbtree

(2) x264 --no-mbtree

x264 (1) --mbtree

(2) x264 --no-mbtree



animation

vertical

animation

vertical

animation

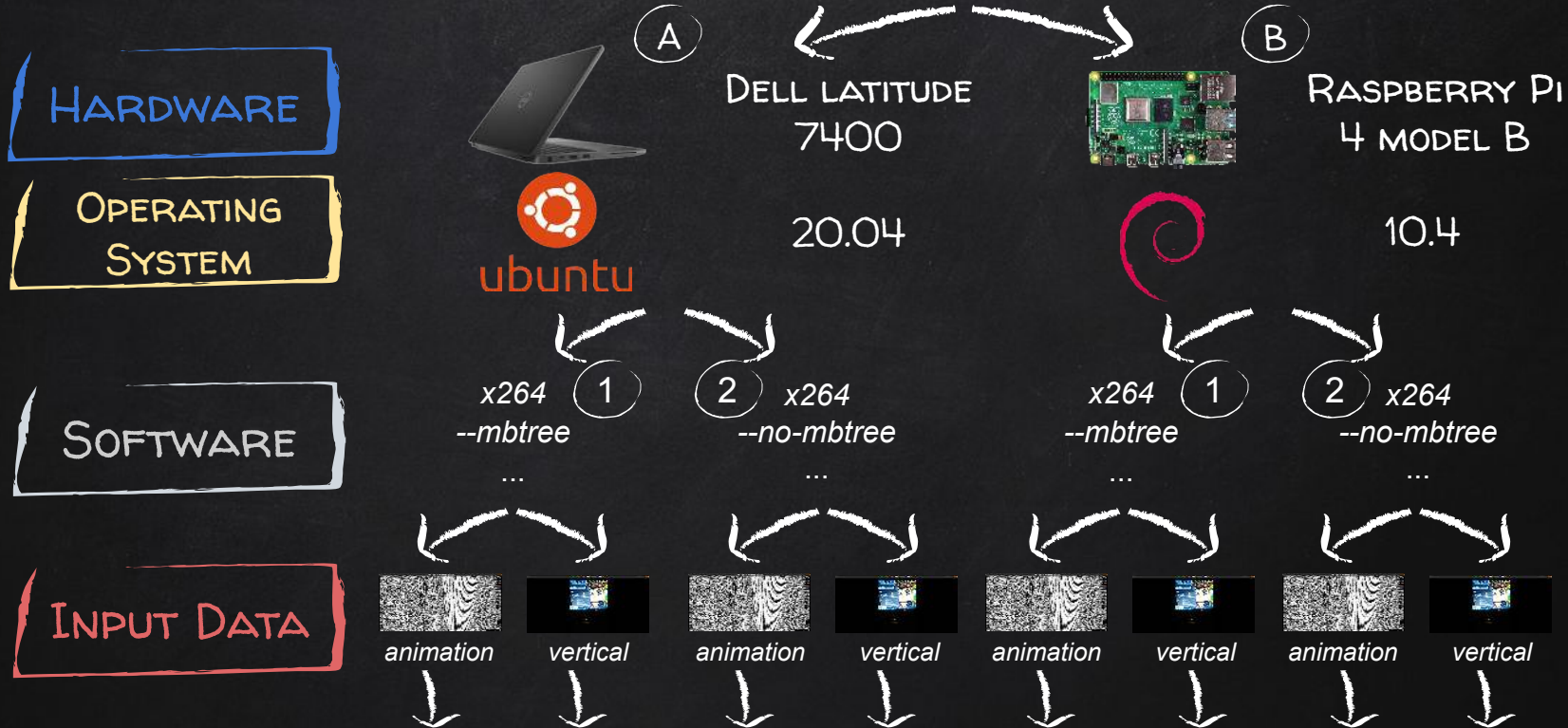
vertical

animation

vertical

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

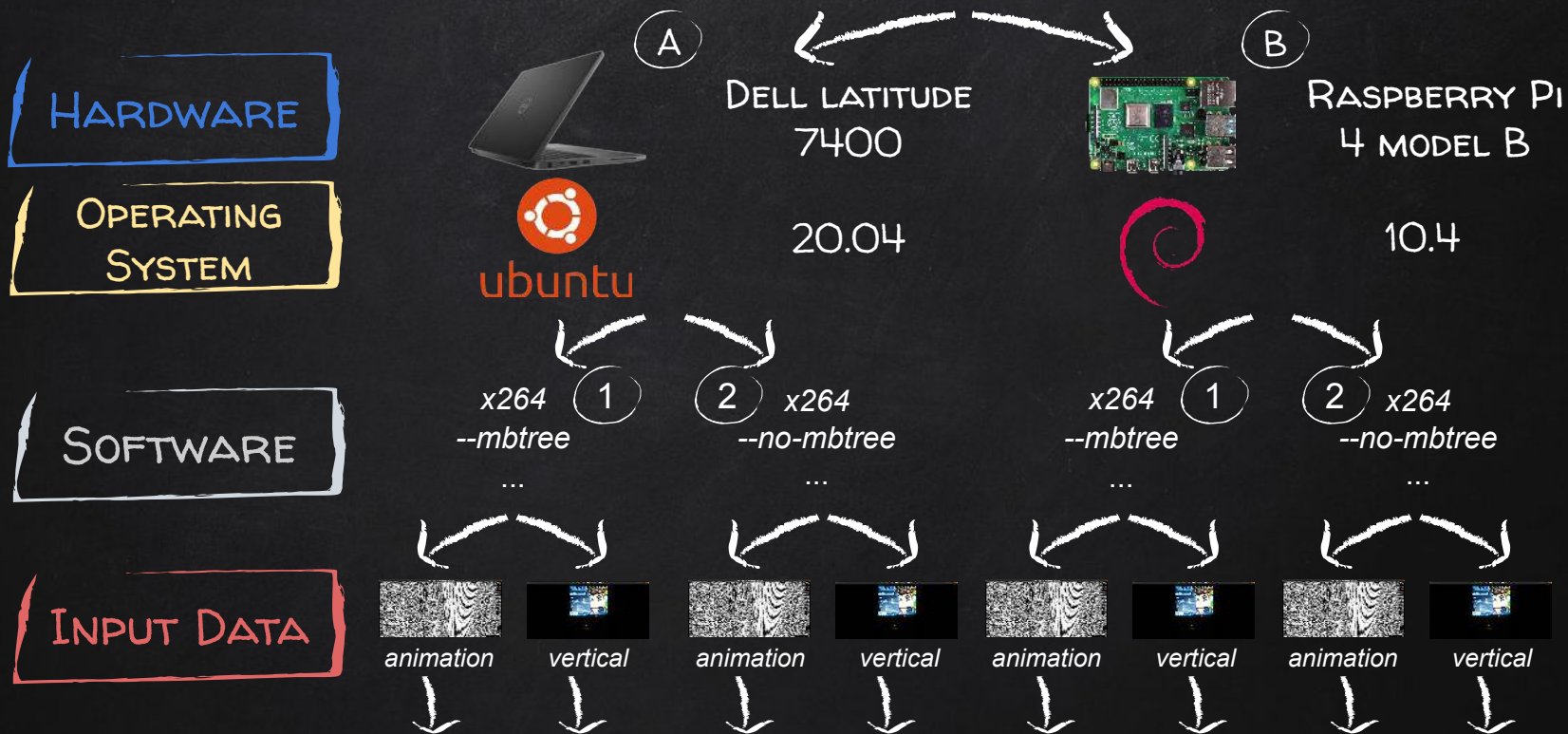
# REAL WORLD EXAMPLE (x264)



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

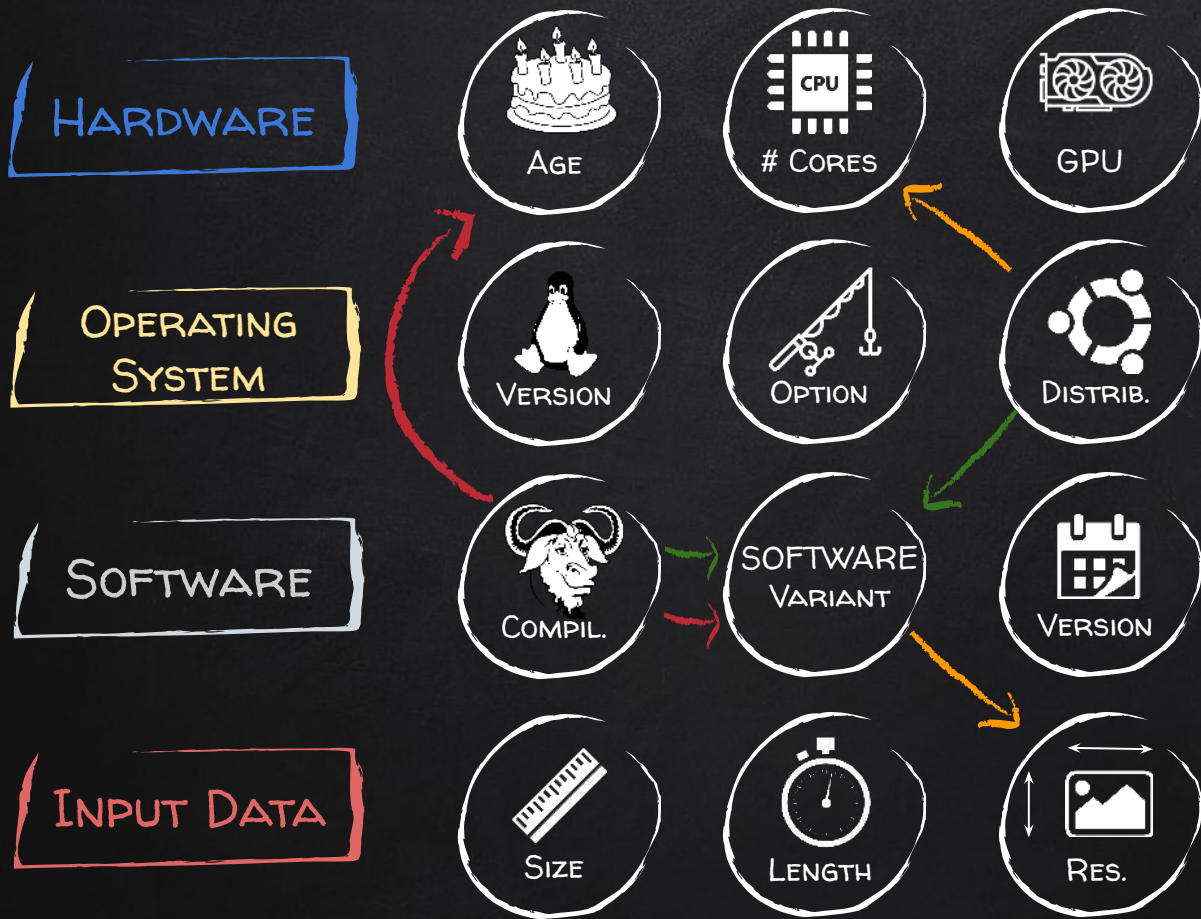


# REAL WORLD EXAMPLE (x264)



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



# 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)
<b>Analysis time</b>	<b>Encoding time</b>	<b>Query time</b>	<b>Execution time</b>
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)
<b>Analysis time</b>	<b>Encoding time</b>	<b>Query time</b>	<b>Execution time</b>
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!**

# 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.





# Do x264 software performances stay consistent across inputs?

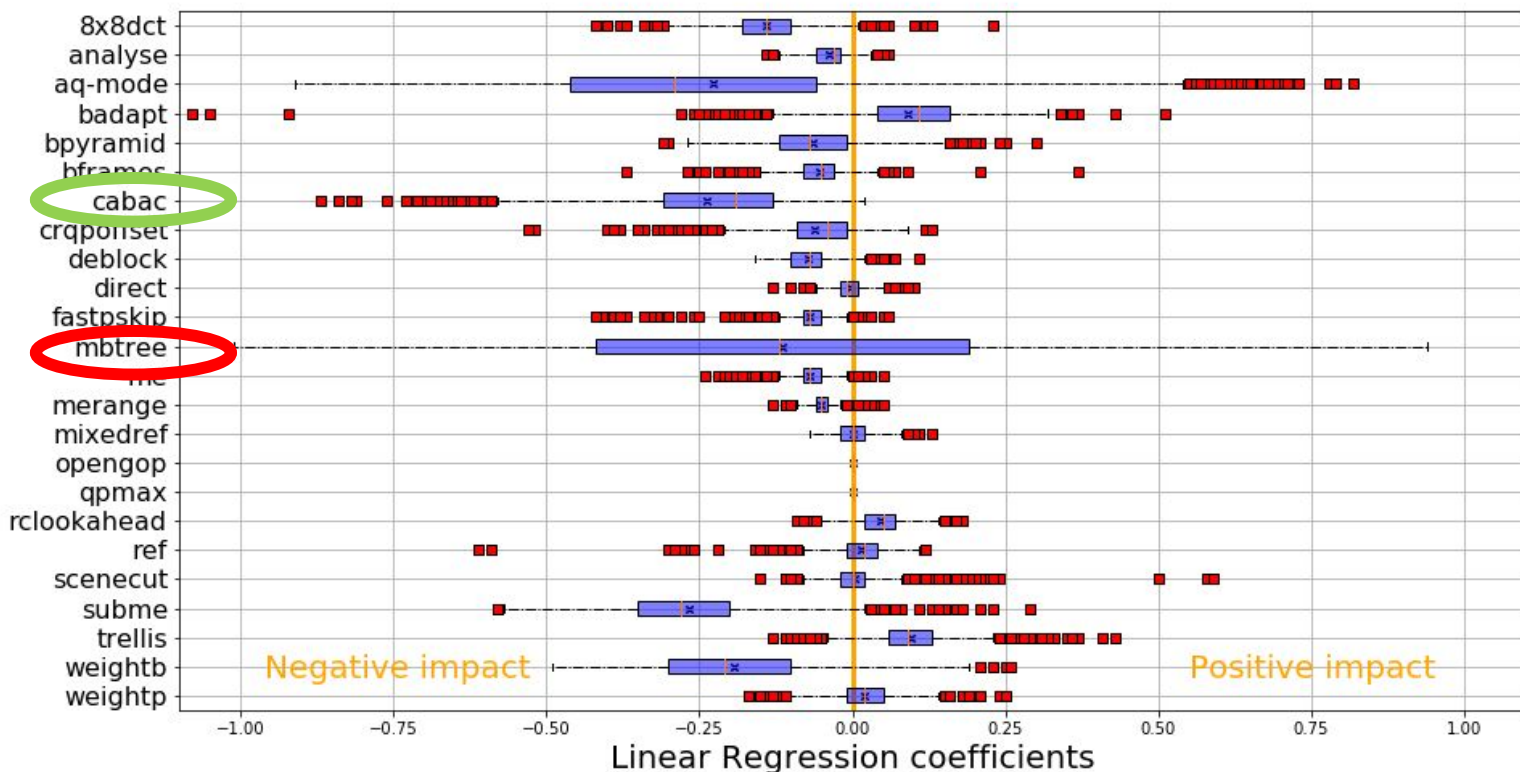


**1397 videos x 201 software configurations**

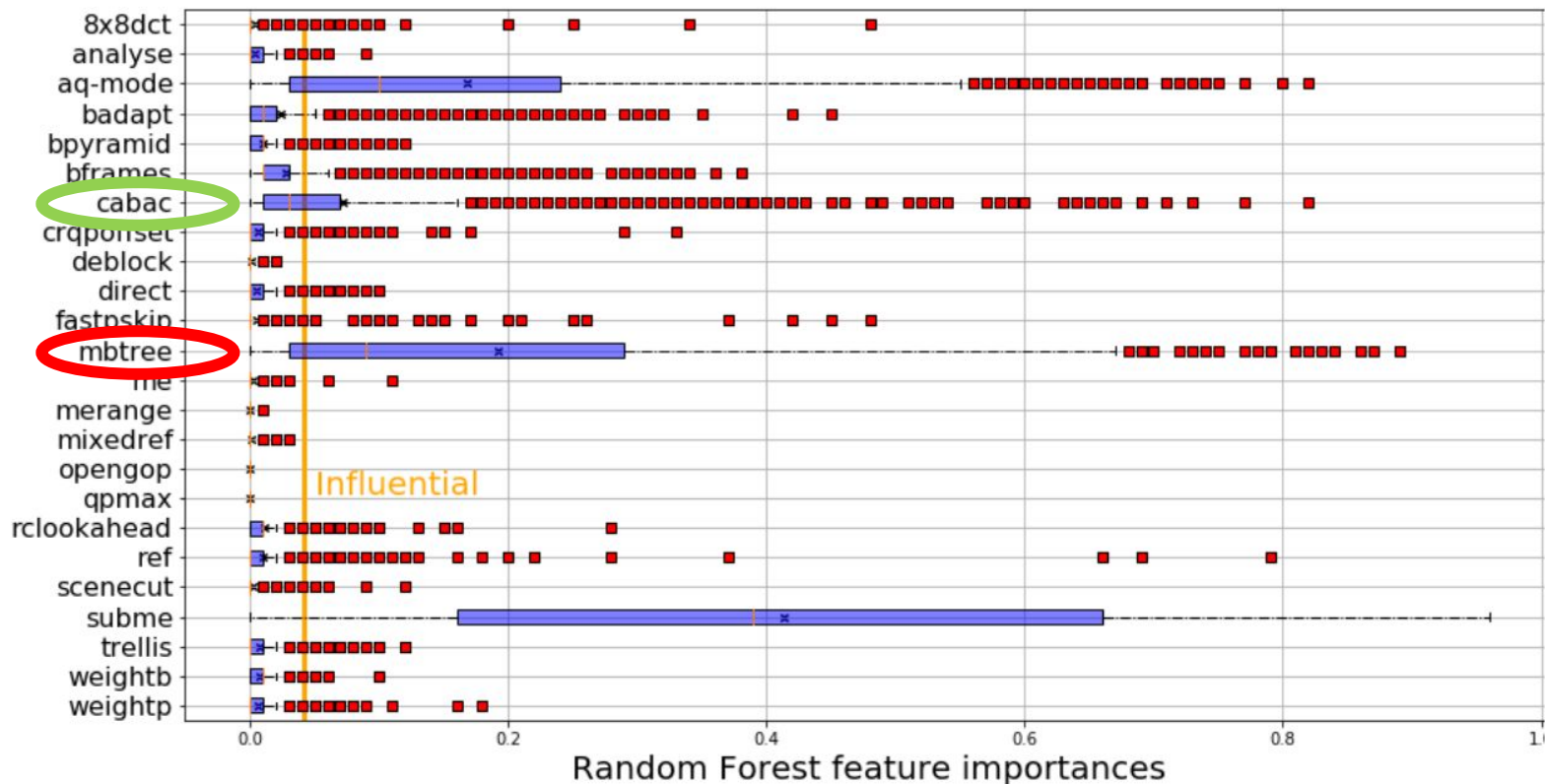
- Encoding time: very strong correlations
  - low input sensitivity
- FPS: very strong correlations
  - low input sensitivity
- CPU usage : moderate correlation, a few negative correlations
  - medium input sensitivity
- Bitrate: medium-low correlation, many negative correlations
  - High input sensitivity
- Encoding size: medium-low correlation, many negative correlations
  - High input sensitivity

two performance models  $f_1$  and  $f_2$      $f_1 = \beta \times f_2 + \alpha$     ?

# Are there some configuration options more sensitive to input videos? (bitrate)



# Are there some configuration options more sensitive to input videos? (bitrate)



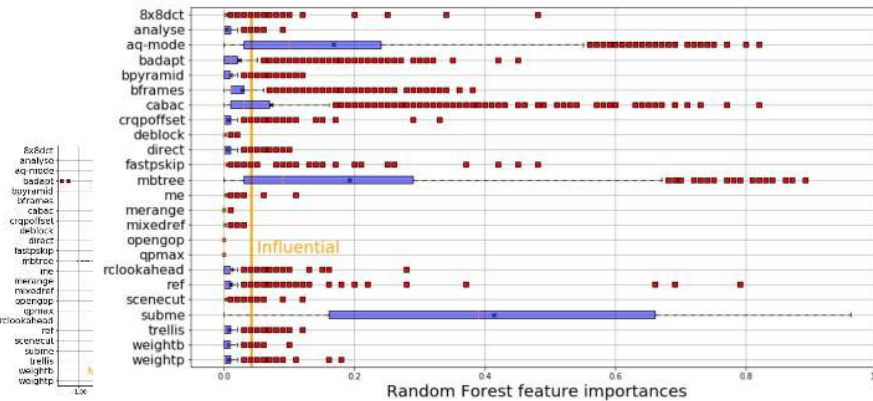
# Practical impacts for users, developers, scientists, and self-adaptive systems

Threats to variability knowledge 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



# 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>



# 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

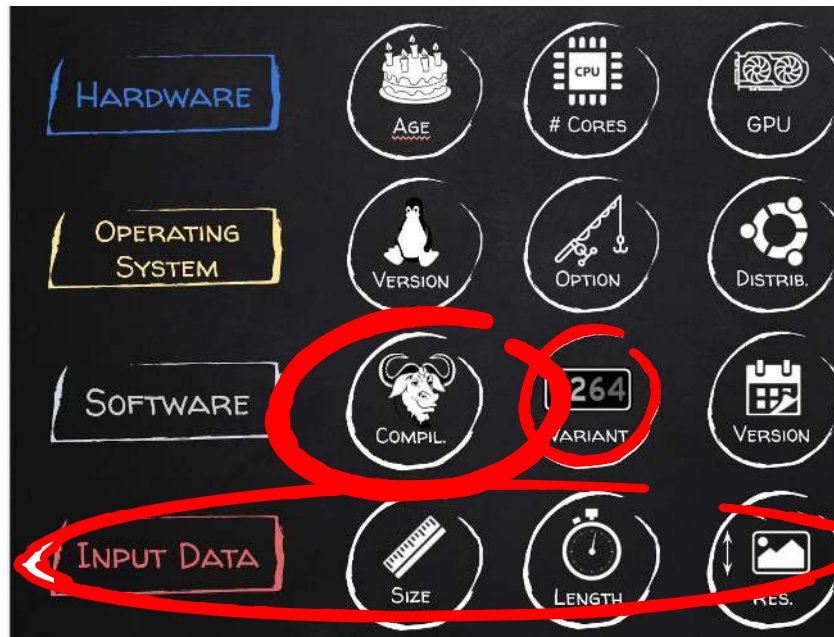
```
--disable-thread      disable multithreaded encoding
--disable-win32thread  disable win32threads (windows only)
--disable-interlaced   disable interlaced encoding support
--bit-depth=BIT_DEPTH set output bit depth (8, 10, all) [all]
--chroma-format=FORMAT output chroma format (400, 420, 422, 444,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

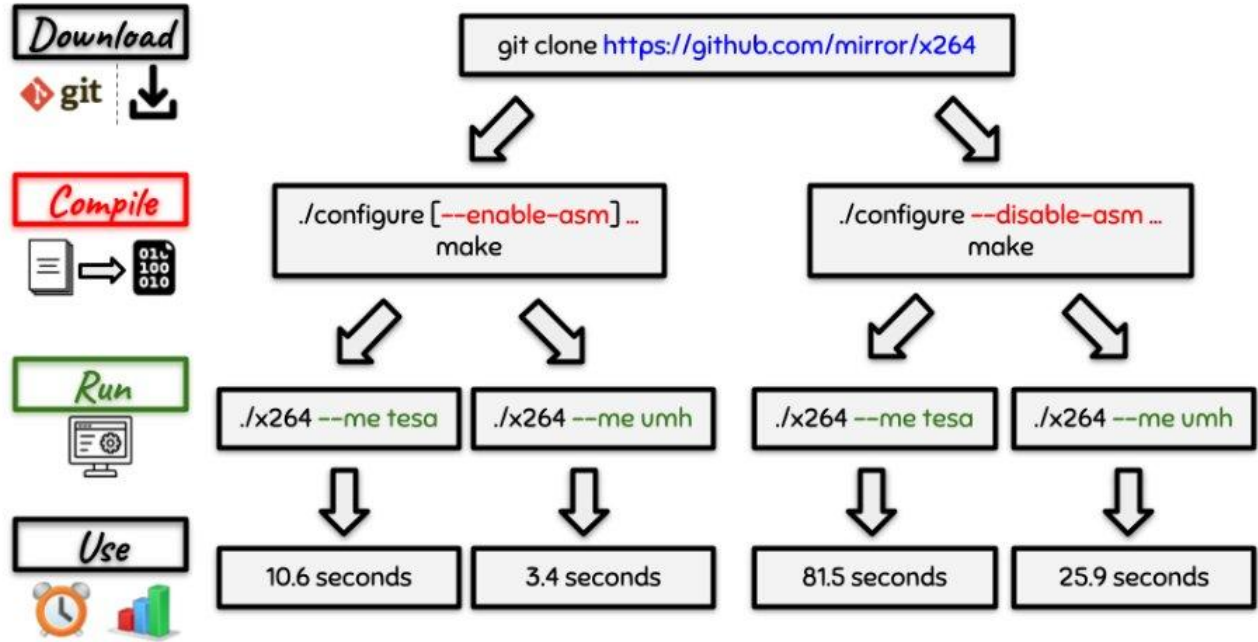
compile-time options



# 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



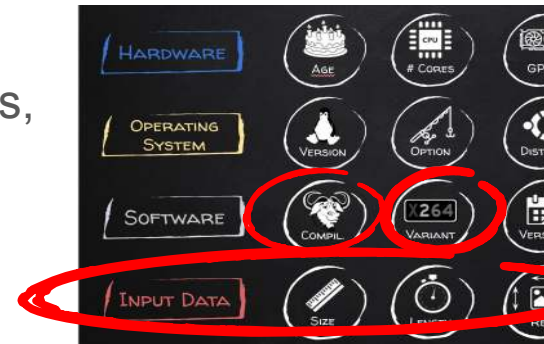
## Key results (for x264)

**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.





# 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!**

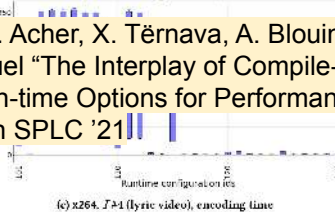
transfer learning of options from one input to another. Users can customize the configuration at run time.

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

vari



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

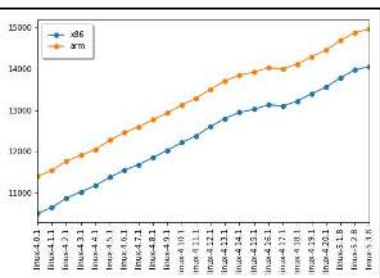


all the input videos, the worst scale of the distribution, they do not interact with the runtime options).



- 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...

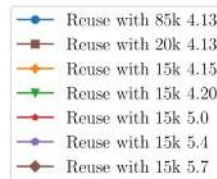
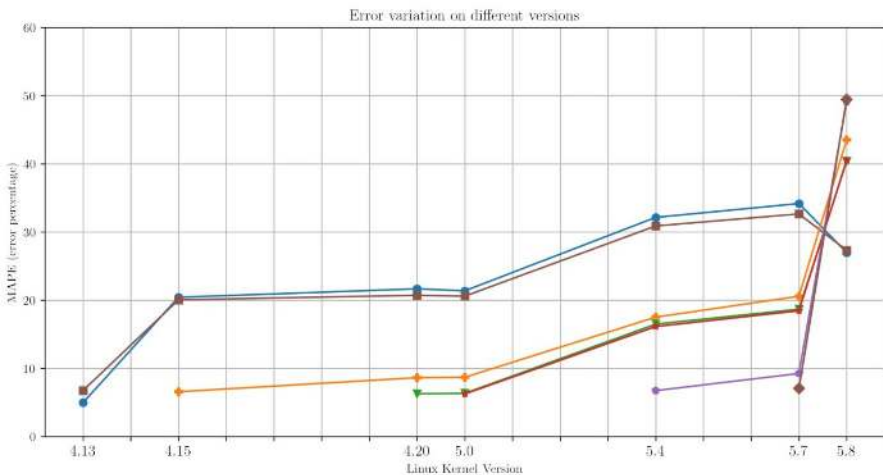




Version	Release Date	LOC	Files	Examples	Seconds/config	Options	Features	Deleted features	New features	$\Delta$ Commits	Files changes
4.13	2017/09/03	16,616,534	60,530	92,562	not available	12,776	9,468	-	-	-	-
4.15	2018/01/28	17,073,368	62,249	39,391	not available	12,998	9,425	342	299	31,052	934,628
4.20	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

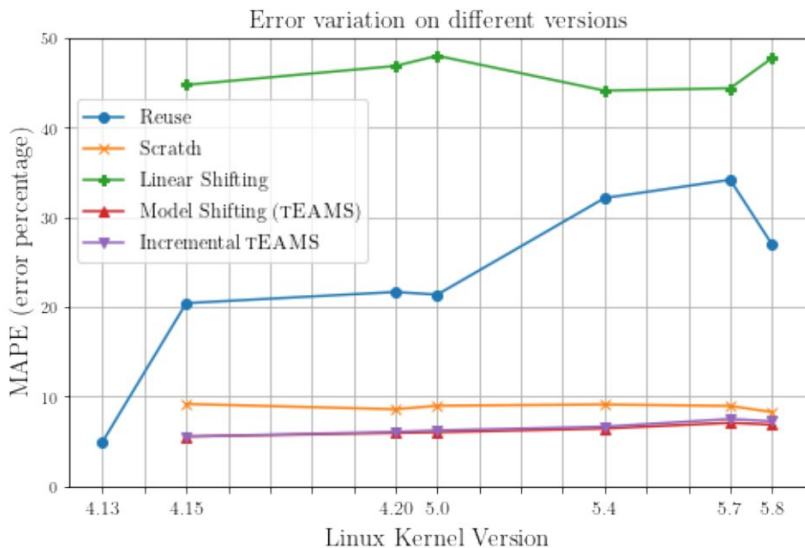
Table I: Dataset properties for each version. The number of deleted/new features, delta commits, files changes are w.r.t. 4.13.

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%**



# Transfer learning to the rescue

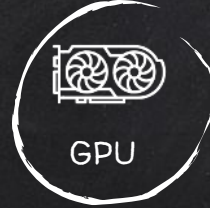
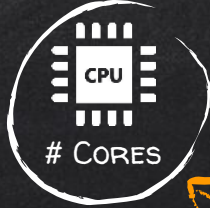
- Mission Impossible: Saving variability knowledge and prediction model 4.13 (15K hours of computation)
- Heterogeneous transfer learning: the feature space is different
- TEAMS: transfer evolution-aware model shifting





# DEEP VARIABILITY

HARDWARE



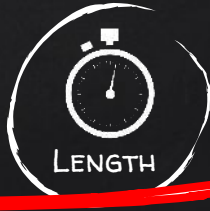
OPERATING SYSTEM



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. ↗

# AGENDA

Reproducible Science and (Deep) Software (Variability)

Deep Software Variability

**Evidence of Deep Software Variability in Computational Science**

Threats and Opportunities

## DEEP VARIABILITY



**It's all about software...  
Where is  
(computational)  
science?**



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

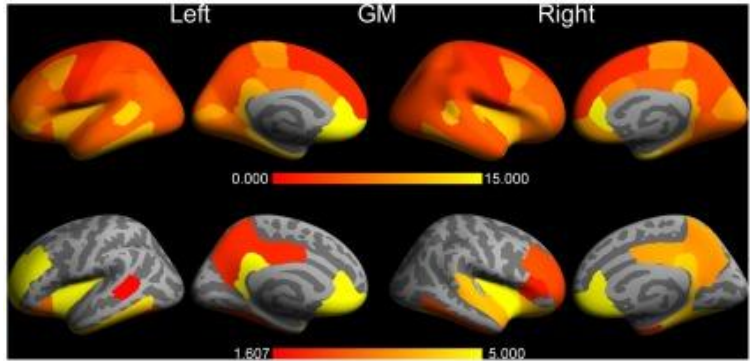
# “Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and

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

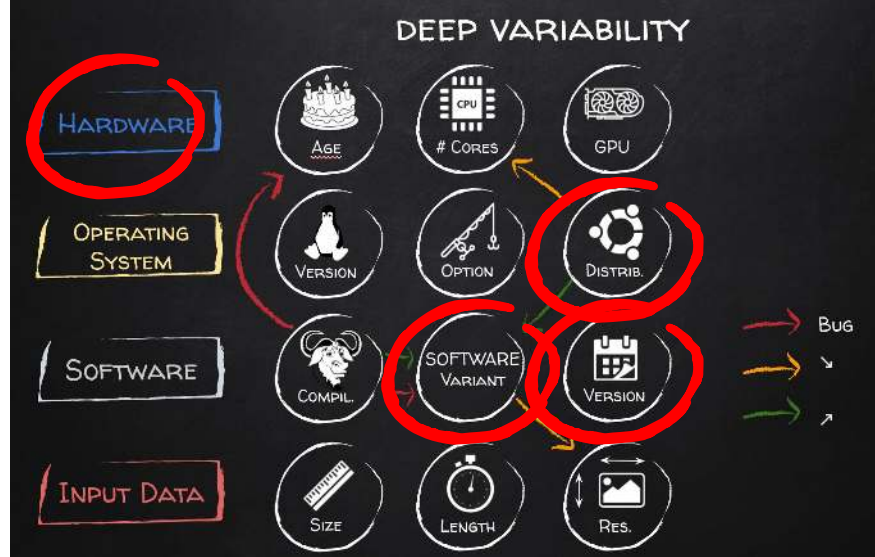
## The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements

Ed H B M Gronenschild <sup>1</sup>, 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.**



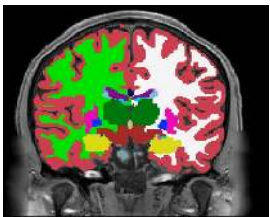
see also Krefting, D., Scheel, M., Freing, A., Specovius, S., Paul, F., and Brandt, A. (2011). “Reliability of quantitative neuroimage analysis using freesurfer in distributed environments,” in *MICCAI Workshop on High-Performance and Distributed Computing for Medical Imaging*. (Toronto, ON).





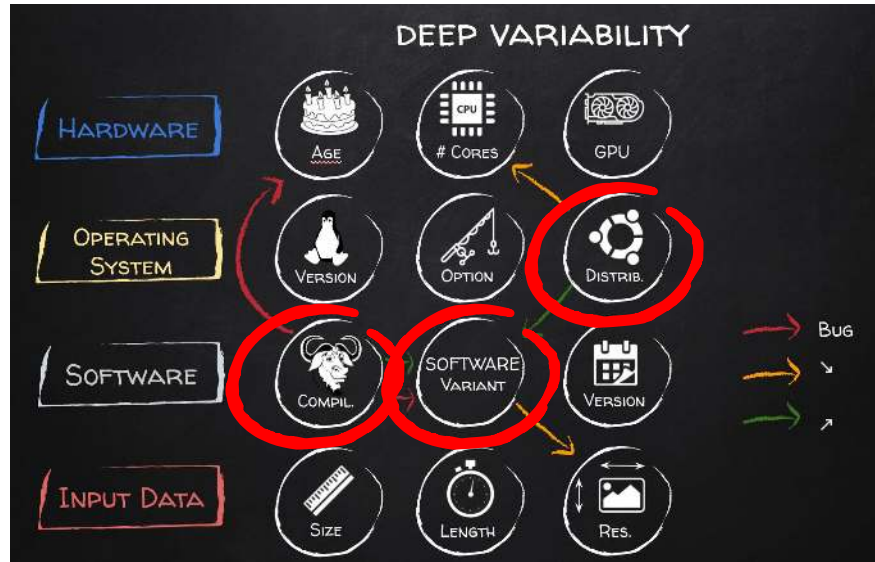
“Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and executed.”

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.

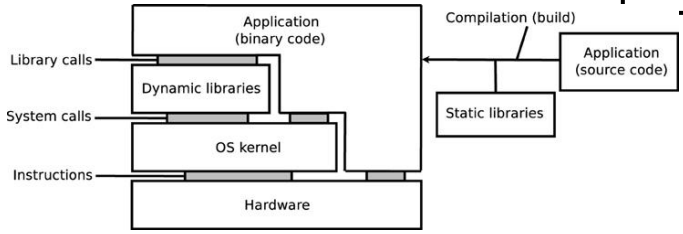
	Cluster A	Cluster B
Applications	Freesurfer 5.3.0, build 1 FSL 5.0.6, build 1 CIVET 1.1.12-UCSF, build 1	Freesurfer 5.3.0, build 1 and 2 FSL 5.0.6, build 1 and 2 CIVET 1.1.12-UCSF, build 1
Interpreters	Python 2.4.3, bash 3.2.25, Perl 5.8.8, tcsh 6.14.00	Python 2.7.5, bash 4.2.47, Perl 5.18.2, tcsh 6.18.01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



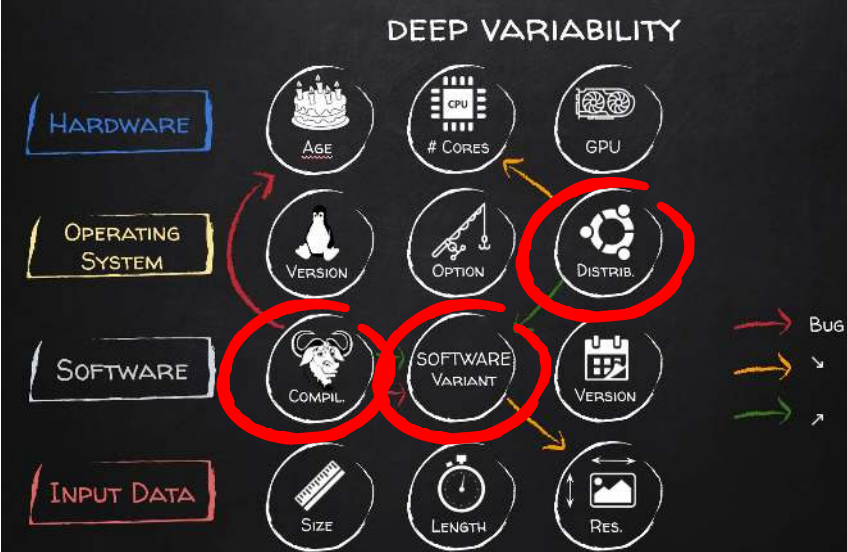
“Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and executed.”

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)



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>middleware</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

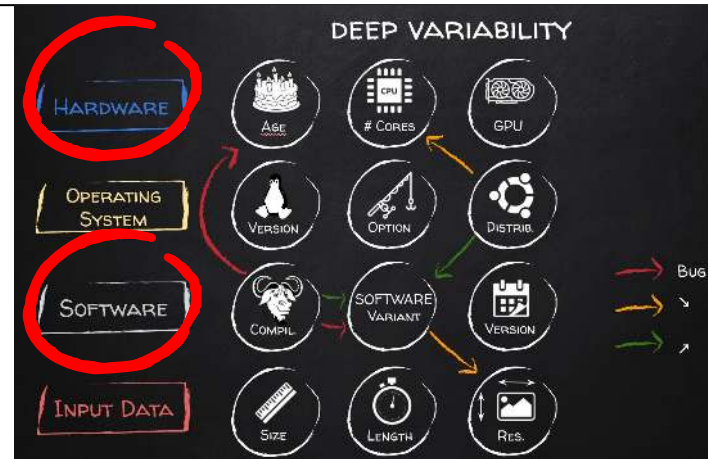
## Association of parameter, software, and hardware variation with large-scale behavior across 57,000 climate models

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>

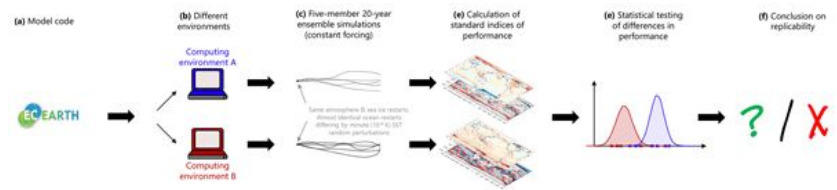
We demonstrate that **effects of parameter, hardware, and software variation are detectable, complex, and interacting**. However, we find most of the effects of parameter variation are caused by a small subset of parameters. Notably, the entrainment coefficient in clouds is associated with 30% of the 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.



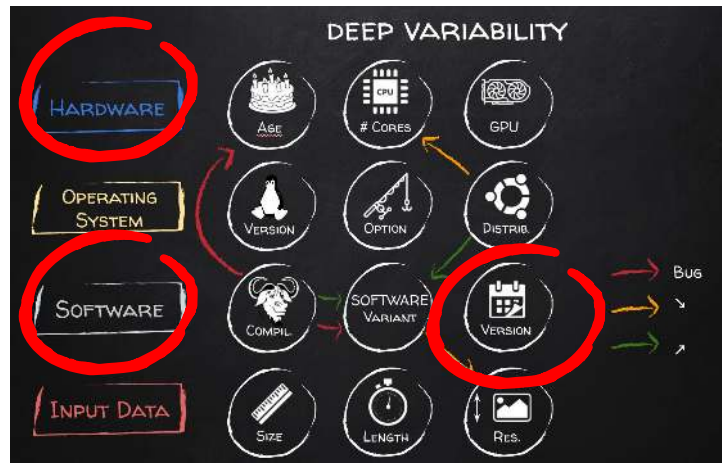
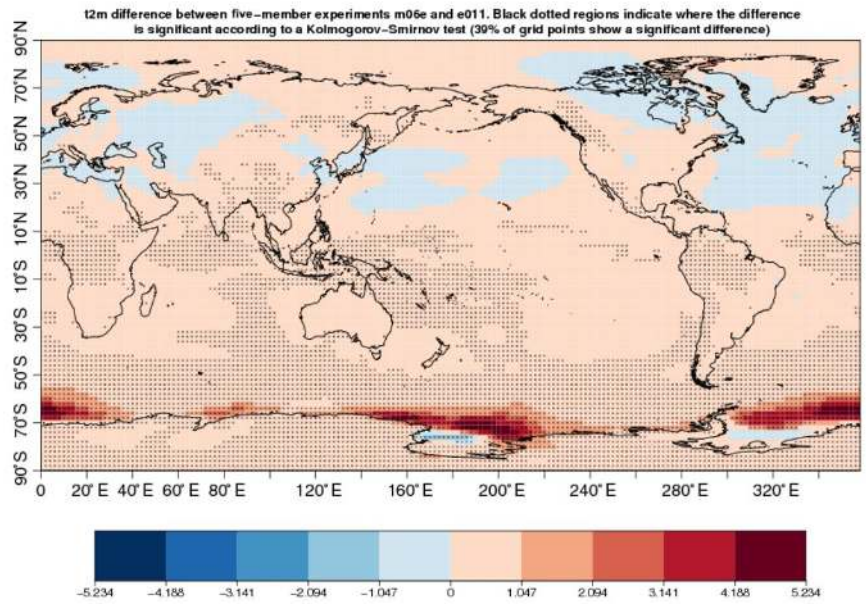
57,067 climate model runs. These runs sample parameter space for 10 parameters with between two and four levels of each, covering 12,487 parameter combinations (24% of possible combinations) and a range of initial conditions

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

François Massonnet<sup>1,2</sup>, Martin Ménégoz<sup>2,3</sup>, Mario Acosta<sup>2</sup>, Xavier Yepes-Arbós<sup>2</sup>, Eleftheria Exarchou<sup>2</sup>, and Francisco J. Doblas-Reyes<sup>2,4</sup>



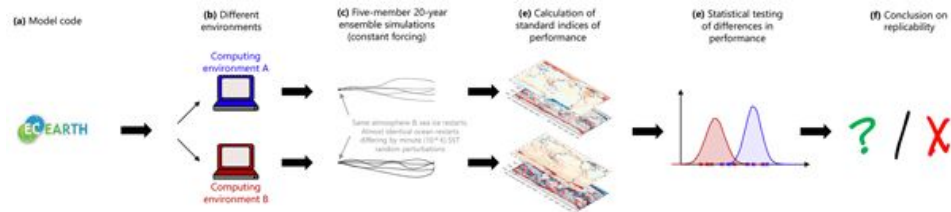
**Can a coupled ESM simulation be restarted from a different machine without causing climate-changing modifications in the results?** Using two versions of EC-Earth: one “non-replicable” case (see below) and one replicable case.





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

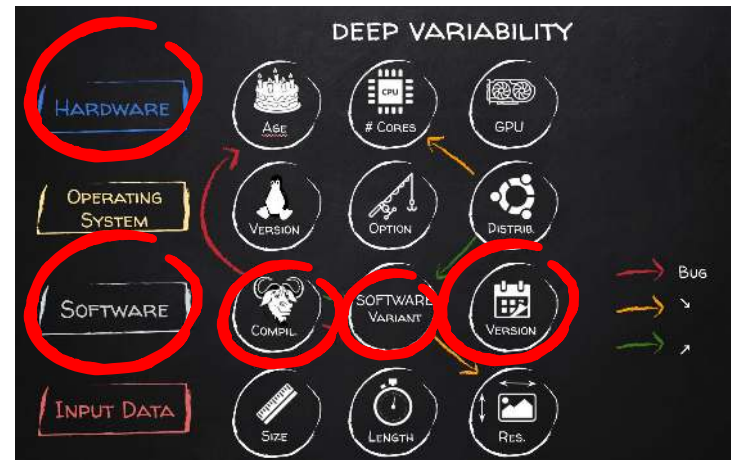
François Massonnet<sup>1,2</sup>, Martin Ménégoz<sup>2,3</sup>, Mario Acosta<sup>1,2</sup>, Xavier Yepes-Arbós<sup>1,2</sup>, Eleftheria Exarchou<sup>1,2</sup>, and Francisco J. Doblas-Reyes<sup>1,2,4</sup>



## 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?).



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<sup>1\*</sup>, Riashat Islam<sup>1,2\*</sup>, Philip Bachman<sup>2</sup>  
Joelle Pineau<sup>1</sup>, Doina Precup<sup>1</sup>, David Meger<sup>1</sup>

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

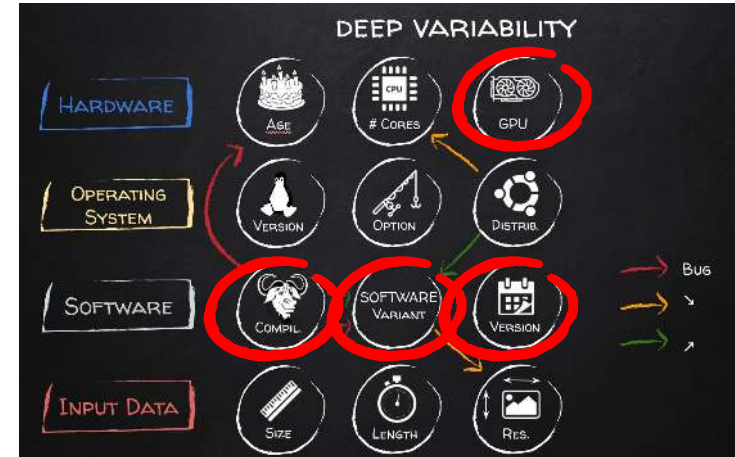
Mechanical and Aerospace Engineering, George Washington University, Washington DC 20052

“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**



# AGENDA

Reproducible Science and (Deep) Software (Variability)

Deep Software Variability

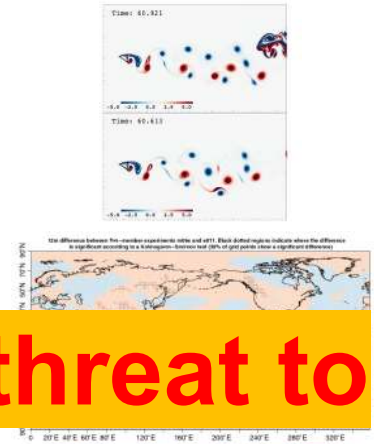
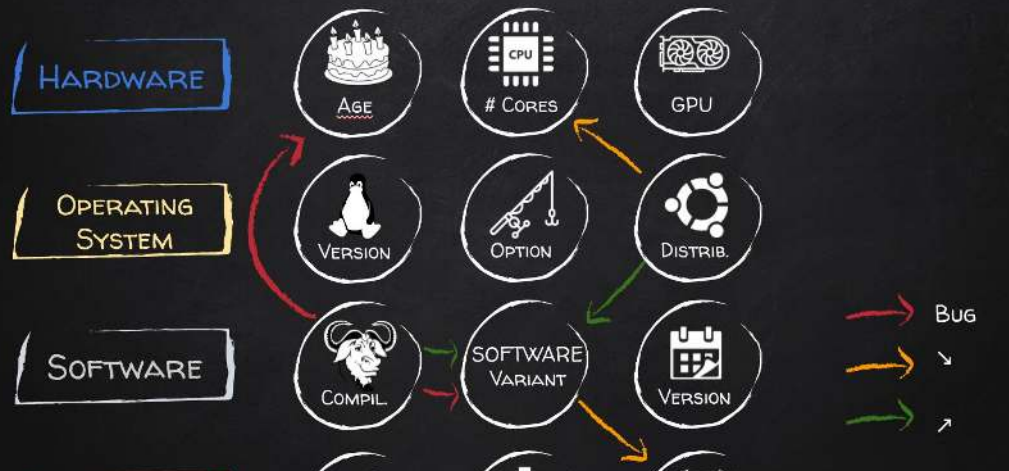
Evidence of Deep Software Variability in Computational Science

**Threats and Opportunities**



# (computational) science

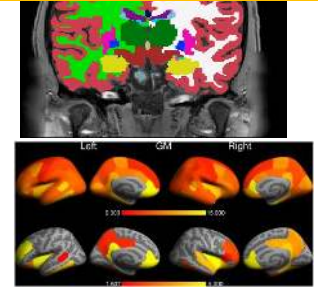
## DEEP VARIABILITY



**Deep software variability is a threat to scientific, software-based experiments**

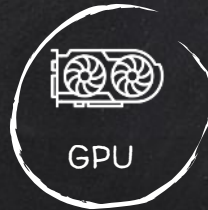
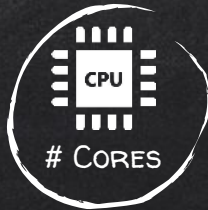


from a set of scripts to automate the deployment to... a comprehensive system containing several features that help researchers exploring various hypotheses



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

HARDWARE



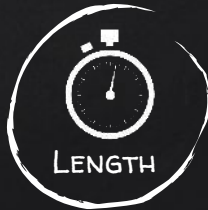
OPERATING SYSTEM



SOFTWARE



INPUT DATA



LIST ALL DETAILS...

AND QUESTIONS:

WHAT IF WE RUN THE EXPERIMENTS ON DIFFERENT:

OS?

VERSION/COMMIT?

PARAMETERS?

INPUT?

# What can we do? (#1 studies)

## Empirical studies about deep software variability

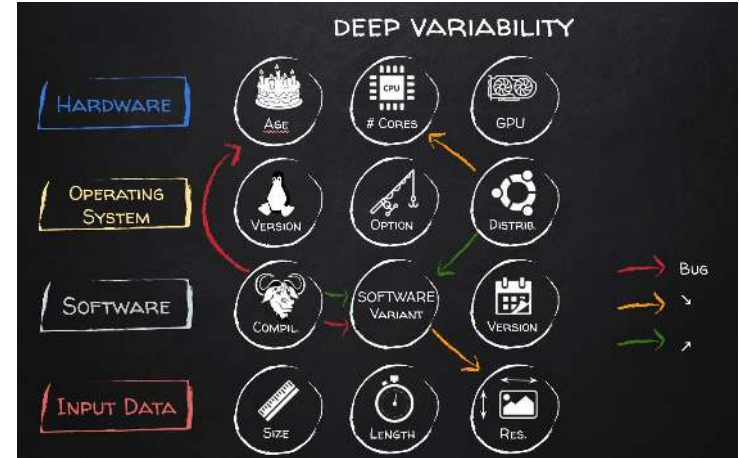
- more subject systems
- more variability layers, including interactions
- more quantitative (e.g., performance) properties

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**

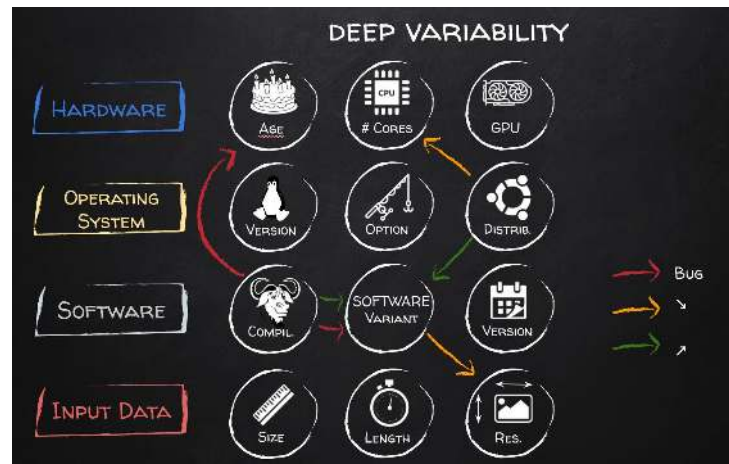


# What can we do? (#2 cost)

## 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
- ...



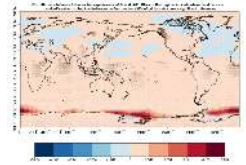
# What can we do? (modelling)

## Modelling variability

- 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
- Challenges:
  - Multiple systems, layers, concerns
  - Different kinds of variability: technical vs domain, accidental vs essential, implicit vs explicit... when to stop modelling?
  - reverse engineering



# What can we do? (robustness)



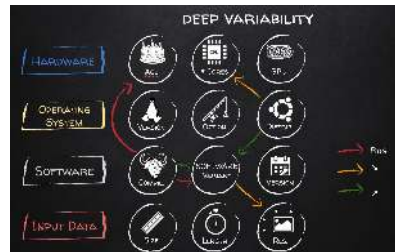
**Robustness (trustworthiness) of scientific results to sources of variability**

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





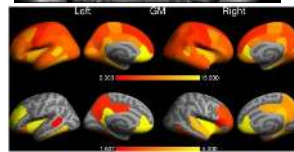
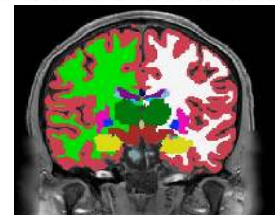
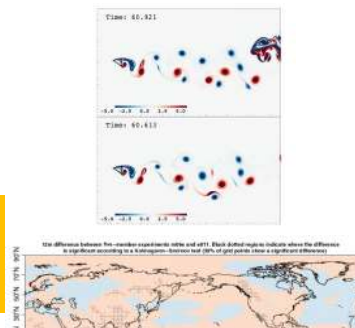
# DEEP VARIABILITY



# (computational) science

**Deep software variability is an opportunity to robustify and augment scientific knowledge**

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





**different data**

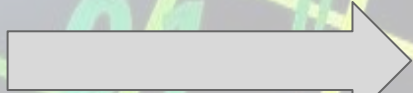


**deep**

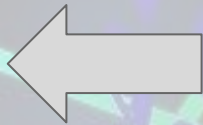
**software**

**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 <sup>1</sup>, Francis Tuerlinckx <sup>1</sup>, Andrew Gelman <sup>2</sup>, Wolf Vanpaemel <sup>3</sup>

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 | Check for updates

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

“When you are a researcher, you want to be the first. Research is about discovery, inventing things that others have not done. It is by definition a form of competition. You have to accept that.” CNRS CEO



At a time of global pandemics, global warming and unprecedented public distrust of politics and democracy, research is anything but an individualistic competition.

Above all, let us value the need for reproducibility in research. Let us prefer the "discovery" that will be verified or even falsified and encourage collaboration.

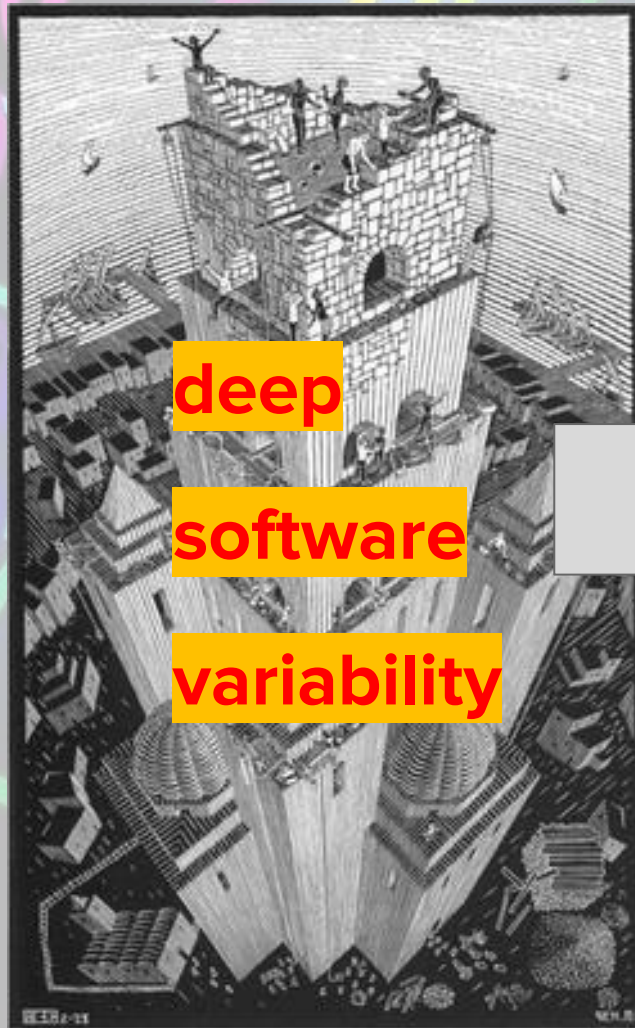
**Deeply exploring the variability space cannot be done alone. Collaborative, distributed effort needed.**



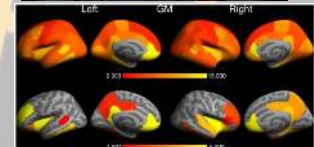
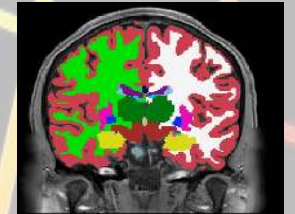
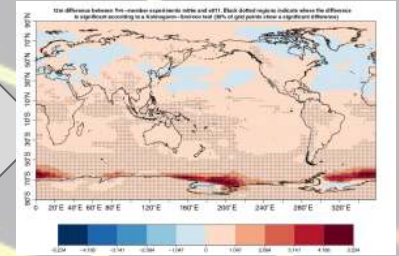
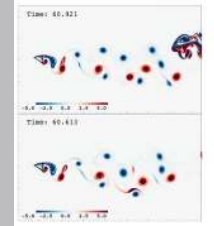
Are we in ivory towers?

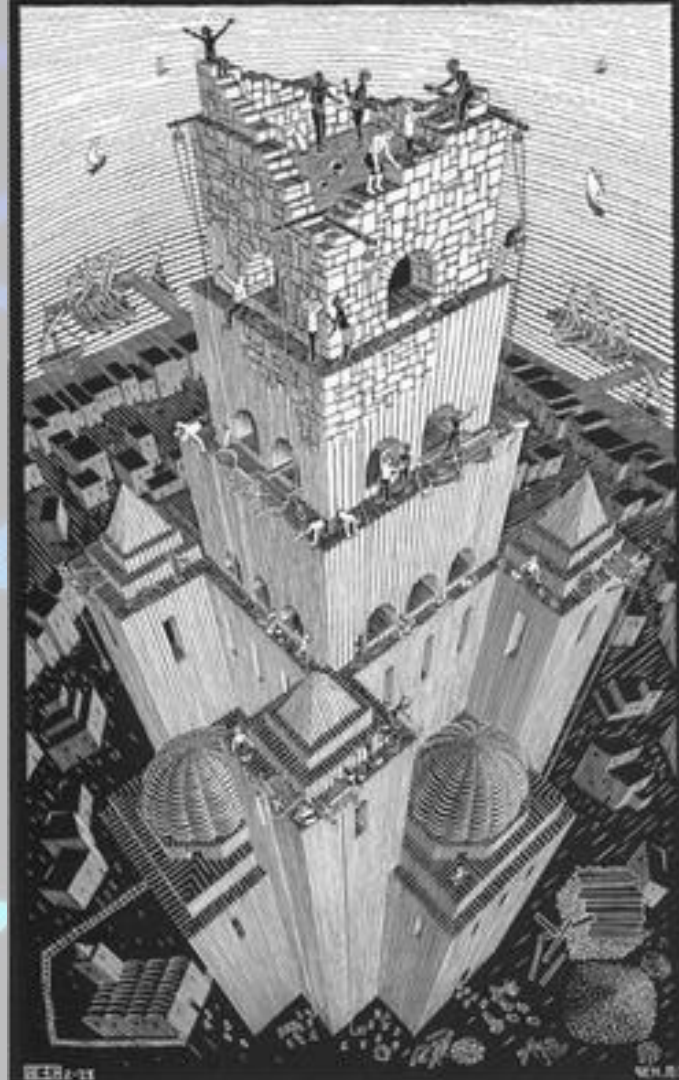
I mean: We're studying software... for the sake of improving software engineering (and it's nice!).

But should we stay?



**(computational)  
science**





**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**

# Reproducibility and replicability: A software engineering perspective

## Practical advices:

- engineer high-quality software;
- automate as much as possible;
- track every decision;
- discuss and recognize threats (deep variability?);
- explore the variants space eg ablation studies

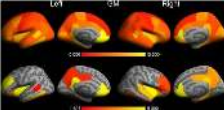
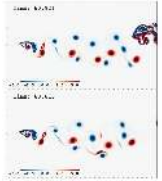
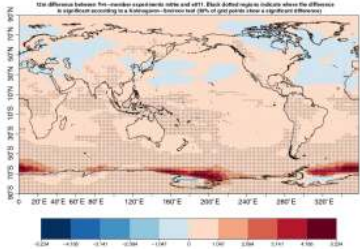
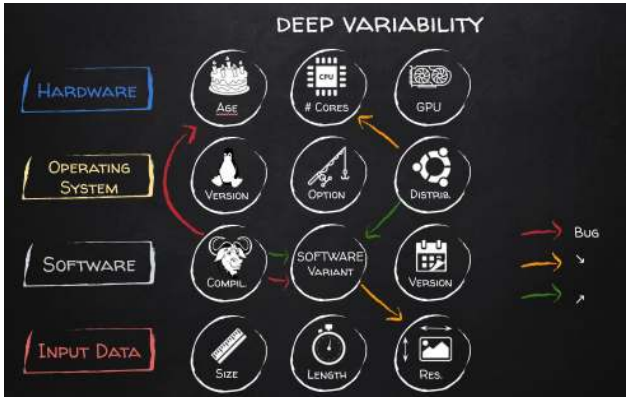
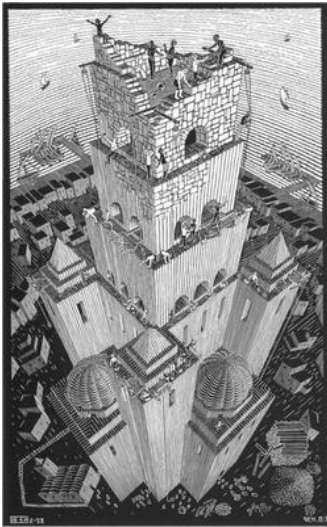
**Deep variability** in your papers/experiments: Are you deep?

No silver bullet... it is an open and difficult problem in science!

DEEP QUESTIONS?



# Reproducible Science and Deep Software Variability







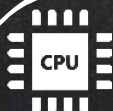
# BACKUP SLIDES

# (BEYOND x264) EMPIRICAL AND FUNDAMENTAL QUESTION

HARDWARE



AGE



# CORES



GPU

HOW DOES DEEP  
SOFTWARE VARIABILITY  
MANIFEST IN THE WILD?

OPERATING  
SYSTEM



VERSION



OPTION



DISTRIB.

SOFTWARE  
SCIENTISTS

SOFTWARE



COMPIL.

VARIANT



VERSION

SHOULD  
OBSERVE THE  
JUNGLE/  
GALAXY!

INPUT DATA



SIZE



LENGTH



RES.

# DEEP SOFTWARE VARIABILITY

## 4 CHALLENGES AND OPPORTUNITIES

Luc Lesoil, Mathieu Acher, Arnaud Blouin, Jean-Marc Jézéquel:

*Deep Software Variability: Towards Handling Cross-Layer Configuration. VaMoS 2021: 10:1-10:8*

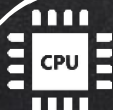
# IDENTIFY THE INFLUENTIAL LAYERS

1

HARDWARE



AGE



# CORES



GPU

OPERATING  
SYSTEM



VERSION



OPTION



DISTRIB.

SOFTWARE



COMPIL.



VARIANT



VERSION

INPUT DATA



SIZE



LENGTH



RES.

PROBLEM  
≠ LAYERS,  
≠ IMPORTANCES ON  
PERFORMANCES

CHALLENGE  
ESTIMATE THEIR EFFECTS

OPPORTUNITY  
LEVERAGE THE USEFUL  
VARIABILITY LAYERS  
& VARIABLES

# TEST & BENCHMARK ENVIRONMENTS

HARDWARE



PROBLEM  
COMBINATORIAL EXPLOSION AND COST

OPERATING SYSTEM



CHALLENGE  
BUILD A  
REPRESENTATIVE, CHEAP  
SET OF ENVIRONMENTS

SOFTWARE



INPUT DATA



OPPORTUNITY  
DIMENSIONALITY REDUCTION



# TRANSFER PERFORMANCES ACROSS ENVIRONMENTS

3



(A)




DELL LATITUDE  
7400




20.04

ubuntu

(B)



RASPBERRY PI  
4 MODEL B



10.4

## PROBLEM

OPTIONS' IMPORTANCES  
CHANGE WITH ENVIRONMENTS

## CHALLENGE

TRANSFER PERFORMANCES  
ACROSS ENVIRONMENTS

## OPPORTUNITY

REDUCE COST OF MEASURE

# CROSS-LAYER TUNING

- BUG
- PERF. ↓
- PERF. ↑

- HARDWARE**
- OPERATING SYSTEM**
- SOFTWARE**
- INPUT DATA**



PROBLEM  
(NEGATIVE) INTERACTIONS  
OF LAYERS

CHALLENGE  
FIND & FIX VALUES TO IMPROVE  
PERFORMANCES

OPPORTUNITY  
SPECIALIZE THE ENVIRONMENT  
FOR A USE CASE

# CHALLENGES FOR DEEP SOFTWARE VARIABILITY

IDENTIFY THE INFLUENTIAL LAYERS

TEST & BENCHMARK ENVIRONMENTS

TRANSFER PERFORMANCES ACROSS ENVIRONMENTS

CROSS-LAYER TUNING

# Wrap-up

Deep software variability is a thing...

miracle and smooth fit of variability layers?

or subtle interactions among layers?

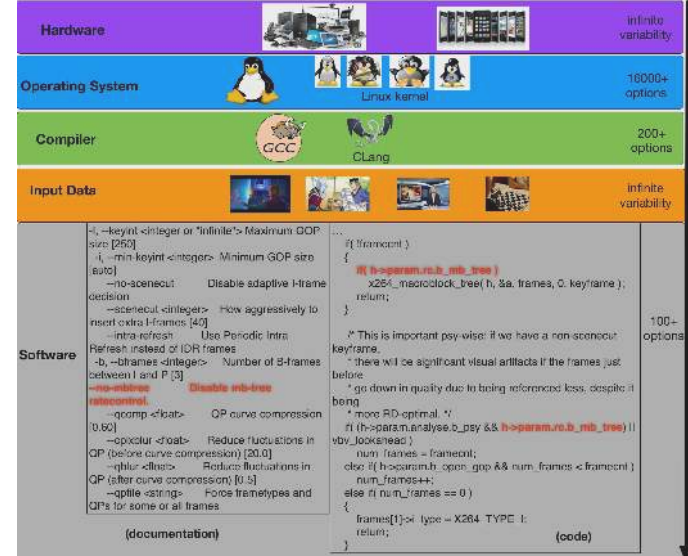
software scientists should **systematically study** the phenomenon

Many challenges and opportunities

**cross-layer tuning** at affordable cost

configuration **knowledge** that generalizes to any context and usage

Dimensionality reduction and **transfer** learning



Variability layers (in time and in space)

# Impacts of deep software variability

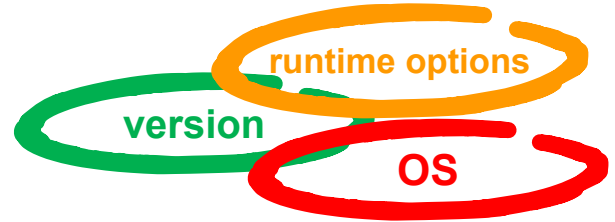
Users/developers/scientists: missed opportunities due to (accidental) variability/complexity

Deep software variability can be seen as a **threat** to knowledge/validity

**Claim: Deep software variability is a threat to scientific, software-based experiments**

Example #1: in the results of neuroimaging studies

- applications of different analysis pipelines (Krefting et al. 2011)
- alterations in software version (Glatard et al. 2015)
- changes in operating system (Gronenschild2012 et al.)



have both shown to cause variation (up to the point it can change the conclusions).

Example #2: on a modest scale, our ICPE 2020 (Pereira et al.) showed that a variation in inputs could change the conclusion about the effectiveness of a sampling strategy for the same software system.

Example #3: I'm reading Zakaria Ournani et al. "Taming Energy Consumption Variations In Systems Benchmarking" ICPE 2020 as an excellent inquiry to control deep variability factors

Example #4: Similar observations have been made in the machine learning community (Henderson et al. 2018)

# Impacts of deep software variability

Users/developers/scientists: missed opportunities due to (accidental) complexity

Deep software variability can be seen as a **threat** to knowledge/validity

Claim: Deep software variability is a threat to scientific, software-based experiments

I propose an exercise that might be slightly disturbing:

**Does deep software variability affect (your|a known) previous scientific, software-based studies?**

(remark: it's mainly how we've investigated deep software variability so far... either as a threat to our own experiments or as a threat identified in papers)



Applications of different analysis pipelines, alterations in software version, and even changes in operating system have both shown to cause variation in the results of a neuroimaging study. Example:

### Reproducibility of neuroimaging analyses across operating systems

Tristan Glatard, Lindsay B. Lewis, Rafael Ferreira da Silva, Roza Adalat, Natacha Beck, Claude Lepage, Pierre Rioux, Marc-Etienne Roussou, Tarek Sherif, Ewa Doolman, Najmeh Khalili-Mahani and Alan C. Evans

Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and executed. We quantify these differences for brain tissue classification, fMRI analysis, and cortical thickness (CT) extraction, using three of the main neuroimaging packages (FSL, Freesurfer and CIVET) and different versions of GNU/Linux. We also identify some causes of these differences using library and system call interception. We find that these packages use mathematical functions based on single-precision floating-point arithmetic whose implementations in operating systems continue to evolve. 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 subcortical tissue classification, fMRI analysis, and cortical thickness extraction. With FSL, most Dice coefficients between subcortical

**DEEP SOFTWARE  
VARIABILITY**

Deep software variability is a threat to scientific, software-based experiments

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 Marne la vallée, France

DAVID.PICARD@ENPC.FR

**DEEP SOFTWARE  
VARIABILITY**

#### 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.



# AGENDA

Interactive course: Reproducible Science and (Deep) Software (Variability)

**Is  $(x+y)+z == x+(y+z)$ ? (TP1)**

A suspicious paper in machine learning... (TP2)

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

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

In Informatics, not in Mathematics

x, y, z: number

Choose the language, the compiler/interpreter, the library, the computing environment, the computer you want...

And tell us!

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

In Informatics, not in Mathematics

x, y, z: number

Choose the programming language, the compiler/interpreter, the library, the computing environment, the computer you want...

Is your work reproducible?



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

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

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

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

write a program that returns a percentage

again: Choose the programming language, the compiler/interpreter, the library, the computing environment, the computer you want...

Code

58 lines (53 loc) · 1.2 KB

```
>
U Unclone
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define XMASK (char)(99)
4 static int randseed=-1;
5
6 /* Macro to return a uniform distributed random real number between 0 and 1 */
7 /* s is the seed, bb the multiplier. 'Maxint &' is used to mask the sign bit*/
8 #define Maxint (0x7fffffff)
9 #define invmaxint 4.656612875e-10
10 #define bb 31415821
11 #ifdef CUSTOM
12 #define uniform (double)(invmaxint*(randseed=(Maxint & ((randseed*bb)-1))))
13 #else
14 #ifdef WIN
15 #define uniform rand()
16 #else
17 #define uniform drand48()
18 #endif
19 #endif
20
21 int associativity_test()
22 {
23     double x = uniform; double y = uniform; double z = uniform;
24     // printf("%f %f %f %d\n",x,y,z,x+(y+z) == (x+y)+z);
25     return x+(y+z) == (x+y)+z;
26 }
27
28 float proportion(int number)
29 {
30     int ok=0;
31     for (int i=0;i<number;i++) ok += associativity_test();
32     return ok*100.0/number;
33 }
34
35
36 /* Here is the entry point of the program */
37 #ifdef OLD_MAIN_C
38 void main(argc, argv)
39     int argc,
40     char **argv;
41 {
42     #else
43     int main(int argc, char **argv) {
44     #endif
45     #ifdef WIN
46     srand(123);
47     #else
48     srand48(123);
```

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

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/>

Please contribute!

new language

new compiler/interpreter

new options/flags

source code

Dockerfile, nix, guix

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

~~Is  $(x+y)+z == x+(y+z)$ ? How often  $(x+y)+z == x+(y+z)$ ?~~

My banker proposed this investment to me:

You give me  $e \approx 2.71828\dots$

The following year, I take 1 euro as a fee and multiply by 1.

The next year, I take 1 euro as a fee and multiply by 2.

The next year, I take 1 euro as a fee and multiply by 3.

...

After  $n$  years, I take 1 euro as a fee and multiply by  $n$ .

To retrieve my money, there is a 1 euro fee.

In 50 years, for my retirement, how much money will I have?

**Write a program that answers this question**



<https://www.college-de-france.fr/fr/agenda/seminaire/seminaires-mecanisees-quand-la-machine-raisonne-sur-ses-langages/arithmetique-des-ordinateurs-et-sa-formalisation>

**Sylvie Boldo talk** (the previous example was based on her talk, thanks!)

Reproduce everything (figures, tables, etc.) that has been said!

Replicate! (change programming language, compiler flags or versions, etc.)

# AGENDA

Interactive course: Reproducible Science and (Deep) Software (Variability)

Is  $(x+y)+z == x+(y+z)$ ? (TP1)

**A suspicious paper in machine learning... (TP2)**

# A suspicious paper in machine learning... (TP2)

Paul (Temple) slides and material here:

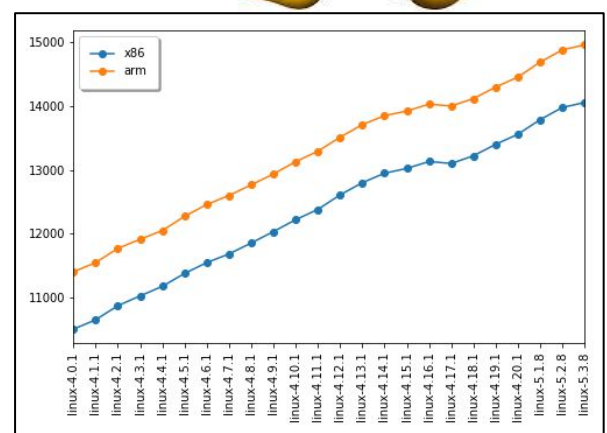
<https://github.com/templep/EJCP2023>



android



15,000+ options



TRISTATE	61.63	3 <sup>9000</sup>
BOOL	36.40	
INT	1.54	2 <sup>6000</sup>
STRING	0.29	
HEX	0.14	

Linux 5.2.8, arm  
(% of types' options)

≈ 10<sup>6000</sup> variants

(without constraints)





VaMoS keynote, 24 february 2022

<https://hal.inria.fr/hal-03528889>

**Abstract** : Biology, medicine, physics, astrophysics, chemistry: all these scientific domains need to process large amount of data with more and more complex software systems. For achieving reproducible science, there are several challenges ahead involving multidisciplinary collaboration and socio-technical innovation with software at the center of the problem. Despite the availability of data and code, several studies report that the same data analyzed with different software can lead to different results. I am seeing this problem as a manifestation of deep software variability: many factors (operating system, third-party libraries, versions, workloads, compile-time options and flags, etc.) themselves subject to variability can alter the results, up to the point it can dramatically change the conclusions of some scientific studies. In this keynote, I argue that deep software variability is a threat and also an opportunity for reproducible science. I first outline some works about (deep) software variability, reporting on preliminary evidence of complex interactions between variability layers. I then link the ongoing works on variability modelling and deep software variability in the quest for reproducible science.

**References** at the ~end of the slides!

## Associative Property of Addition

**You:** Is  $(x+y)+z == x+(y+z)$  in informatics?

**ChatGPT:** Yes, in most programming languages, the statement  $(x + y) + z == x + (y + z)$  holds true due to the associative property of addition. This property states that the way numbers are grouped doesn't change the sum, which applies both in mathematics and informatics.

However, keep in mind that in the case of floating point numbers, because of the way they are represented internally, the order of operations can sometimes impact the result due to round-off errors. This is more the exception than the rule, and in general, the property will hold true.

**You:** Give me an example with floating point numbers

**ChatGPT:** Sure, I can give you an example in Python where floating point precision may make a difference due to the order of operations. However, please remember that these cases are rare and generally the order won't matter with addition. The issue arises from the fact that computers cannot precisely represent some decimal values.

```
# Python code
x = 1.0
y = 1e-16 # 0.0000000000000001
z = -1.0

# Add in one order
first_order = (x + y) + z
print(first_order) # prints: 1e-16

# Add in another order
second_order = x + (y + z)
print(second_order) # prints: 0.0
```

In this example,  $(x + y) + z$  is not equal to  $x + (y + z)$  due to floating point precision errors. In the first case,  $x$  and  $y$  are added first, resulting in a value slightly greater than 1.0. Then  $z$  is added, yielding a small positive number ( $1e-16$ ). In the second case,  $y$  and  $z$  are added first, which exactly cancel each other out because they are precisely opposite in value. Then  $x$  is added to zero, yielding 1.0.

It's crucial to note that this discrepancy arises because of the limitations of floating-point precision and not a failure of the associative property.

**You:** Is it true for any programming language?



**ChatGPT:** The behavior regarding floating point precision is consistent across virtually all modern programming languages because they typically use the same underlying standard for floating point numbers: the IEEE 754 standard.

So, yes, the precision issues you might encounter in Python with floating point numbers can also appear in other languages like C++, Java, JavaScript, and so on. This is because the inaccuracies originate from the way computers represent floating point numbers, not from the programming languages themselves.

That said, there might be some differences in how each language handles these precision issues, and some languages provide ways to mitigate precision errors. For instance, some languages like Python have built-in libraries for arbitrary-precision arithmetic, or you might use fixed-point arithmetic in situations where it's crucial to maintain a certain level of precision.

**You:** What about Rust?

**ChatGPT:** Rust, like other modern programming languages, follows the IEEE 754 standard for floating point arithmetic. This means that the precision issues that can arise with floating point numbers in languages like Python, C++, Java, and JavaScript can also occur in Rust.

Here is an example in Rust similar to the Python example above:

```
fn main() {
    let x = 1.0f64;
    let y = 1e-16f64;
    let z = -1.0f64;

    let first_order = (x + y) + z;
    println!("{}", first_order); // prints: 1e-16

    let second_order = x + (y + z);
    println!("{}", second_order); // prints: 0.0
}
```

This Rust code will have the same outcome as the Python code.  $(x + y) + z$  does not equal  $x + (y + z)$  due to floating point precision errors. Again, it's important to note that this is due to the limitations of floating point representation and not a failure of the associative property.

**You:** what about C?

**ChatGPT:** Yes, the same floating point precision issue would occur in the C programming language as well. Here's the equivalent code to the previous examples:

```

#include <stdio.h>

int main() {
    double x = 1.0;
    double y = 1e-16;
    double z = -1.0;

    double first_order = (x + y) + z;
    printf("%e\n", first_order); // prints: 1.000000e-16

    double second_order = x + (y + z);
    printf("%e\n", second_order); // prints: 0.000000e+00

    return 0;
}

```

This code will behave just like the Python and Rust examples, with  $(x + y) + z$  not equalling  $x + (y + z)$  due to floating point precision errors. Just like in other languages, this issue arises from how computers represent floating point numbers, and not from a failure of the associative property.

**You:** replace double by float?

**ChatGPT:** In C, a float has less precision than a double. The double type has more bits of precision than the float type, so the effects of floating-point precision issues can be even more noticeable with float. Here's an example:

```

#include <stdio.h>

int main() {
    float x = 1.0f;
    float y = 1e-16f;
    float z = -1.0f;

    float first_order = (x + y) + z;
    printf("%e\n", first_order); // prints: 0.000000e+00

    float second_order = x + (y + z);
    printf("%e\n", second_order); // prints: 0.000000e+00

    return 0;
}

```

In this case, you'll notice that both `first_order` and `second_order` print `0.000000e+00`. This happens because `1e-16` is too small to be represented accurately using a float, so it gets rounded down to `0.0f`. Therefore, both  $(x + y) + z$  and  $x + (y + z)$  end up as `0.0f`. So, in this specific case, the two

expressions are equal, but not for the reason you might expect; it's due to lack of precision in the `float` type rather than the associative property of addition.

**You:** using `float` in C, can you devise a counterexample of  $(x+y)+z == x+(y+z)$

**ChatGPT:** Certainly! When using `float` in C, the effects of precision issues are more noticeable due to its lower precision compared to `double`. Here's an example where  $(x + y) + z \neq x + (y + z)$  due to precision errors:

```
#include <stdio.h>

int main() {
    float x = 1.0e10f;
    float y = 1.0f;
    float z = -1.0e10f;

    float first_order = (x + y) + z;
    printf("%f\n", first_order); // prints: 1.000000

    float second_order = x + (y + z);
    printf("%f\n", second_order); // prints: 0.000000

    return 0;
}
```

In this case, the precision of `float` is not sufficient to hold the result of  $x + y$  accurately because  $y$  is significantly smaller than  $x$ . Therefore, the result of  $(x + y) + z$  is  $1.0$  due to the loss of precision in  $x + y$ . On the other hand,  $y + z$  equals  $-1.0e10$  exactly because  $y$  and  $z$  are of comparable magnitude, and then adding  $x$  results in  $0.0$ . So,  $(x + y) + z$  is not equal to  $x + (y + z)$ .

**You:** using `double` in the C program, any chance to mitigate the error in precision with compiler flags?

**ChatGPT:** While there may be compiler flags to control floating-point behavior, they typically won't solve the issue at hand. The precision problem isn't something a compiler can easily fix; it's a fundamental limitation of the floating-point number system used in most computer hardware.

For instance, the GCC and Clang compilers offer some flags to control floating point behavior (like `-ffloat-store`, `-fexcess-precision`, `-frounding-math`, etc.). However, these flags mainly affect the way intermediate results are stored or how rounding is handled. They are not capable of increasing the precision of `double` beyond its inherent limits according to the IEEE 754 standard.

If you need more precision than `double` can provide and want to avoid these types of errors, you might need to use libraries for arbitrary precision arithmetic, such as GMP or MPFR, but these will likely make your computations slower. Another approach is to rearrange your equations so that you avoid subtracting nearly equal numbers or adding a very small number to a very large one, as these are the types of operations that tend to lead to significant loss of precision.

**You:** what about `-mfpmath=387` ?

**ChatGPT:** The `-mfpmath=387` option is specific to the GCC compiler on x86 platforms. It instructs the compiler to use the floating-point unit on the x87 coprocessor for floating-point calculations instead of the SSE/SSE2 units.

The x87's FPU operates at a higher internal precision (80 bits) than the usual `double` (which is 64 bits). So, theoretically, computations done using the x87's FPU can be more accurate because of the extra precision during intermediate calculations.

However, the difference in precision between the x87 FPU and SSE/SSE2 only matters for the intermediate calculations. The final stored result will still be truncated to fit into a `double` or `float`, so it doesn't inherently increase the precision of your final results.

Moreover, the use of `-mfpmath=387` may not guarantee that the rounding errors would be eliminated completely. In complex calculations, the final results may still suffer from floating point precision issues due to the inherent limitations of the floating-point representation.

In summary, while `-mfpmath=387` might provide slightly more accurate results in some specific cases due to higher precision in intermediate steps, it doesn't completely mitigate the limitations of the floating-point number system. If absolute precision is required, consider using libraries that support arbitrary-precision arithmetic, or apply numerical methods that are designed to minimize errors.

**You:** what about clang?

**ChatGPT:** The Clang compiler, unlike GCC, does not currently support the `-mfpmath` option (as of my knowledge cutoff in September 2021).

Clang and GCC are designed to be mostly command-line compatible, so for many options, if GCC has a flag, Clang is likely to have the same or a similar one. However, this is not the case for `-mfpmath`. The main reason is that `-mfpmath=387` is a very hardware-specific flag tied to the x87 floating point unit, which is not used by Clang for floating point computations by default. Clang typically uses SSE for floating point arithmetic on x86 and x86\_64.

As a result, if you're using Clang and you need to increase the precision of your floating point calculations, you'll need to explore other options. You might want to consider using a library for arbitrary precision arithmetic or a library for multiprecision floating-point computations, such as MPFR, or changing the way you perform your calculations to minimize the impact of rounding errors.

It's always best to check the most recent compiler documentation, as compiler features and flags can change over time.

**You:** can you adapt the float example with GMP?

**ChatGPT:** The GNU Multiple Precision Arithmetic Library (GMP) provides functions for multiple precision arithmetic. However, GMP mainly provides functions for integer and rational number calculations, not floating-point calculations.

If you need more precision for floating-point calculations, the GNU MPFR library (Multiple Precision Floating-Point Reliable) is a good choice. It is built on top of GMP and provides functions for high-precision floating-point arithmetic.

Here is an example of how you might use MPFR to calculate the expression in a way that demonstrates the associative property of addition:

```
#include <stdio.h>
#include <mpfr.h>

int main() {
    mpfr_t x, y, z, first_order, second_order, temp;

    // Initialize variables
    mpfr_inits2(128, x, y, z, first_order, second_order, temp, (mpfr_ptr) 0);

    mpfr_set_d(x, 1.0e10, MPFR_RNDN);
    mpfr_set_d(y, 1.0, MPFR_RNDN);
    mpfr_set_d(z, -1.0e10, MPFR_RNDN);

    // (x + y) + z
    mpfr_add(temp, x, y, MPFR_RNDN);
    mpfr_add(first_order, temp, z, MPFR_RNDN);

    // x + (y + z)
    mpfr_add(temp, y, z, MPFR_RNDN);
    mpfr_add(second_order, x, temp, MPFR_RNDN);

    // Print results
    mpfr_printf("(x + y) + z = %.15Rf\n", first_order);
    mpfr_printf("x + (y + z) = %.15Rf\n", second_order);
}
```

```

    // Free memory
    mpfr_clears(x, y, z, first_order, second_order, temp, (mpfr_ptr) 0);

    return 0;
}

```

In this example, we've used the MPFR library to perform the calculations with a precision of 128 bits (which is significantly higher than the precision of a `float` or a `double`). As a result, you'll see that  $(x + y) + z$  equals  $x + (y + z)$ , demonstrating the associative property of addition.

Remember that to compile this code, you'll need to link against the MPFR and GMP libraries. You can do this by adding `-lmpfr -lgmp` to your compile command. For example:

```
gcc filename.c -o output -lmpfr -lgmp
```

Please note that using high-precision libraries like MPFR can be slower and more memory-intensive than using regular floating-point types. These libraries should be used when you really need the extra precision, and not for general, everyday calculations.

**You:** what about Java?

**ChatGPT:** Java provides a class named `BigDecimal` in `java.math` package for high precision calculations. Here's how you might use `BigDecimal` to calculate the same expression:

```

import java.math.BigDecimal;

public class Main {
    public static void main(String[] args) {
        BigDecimal x = new BigDecimal("1E10");
        BigDecimal y = new BigDecimal("1");
        BigDecimal z = new BigDecimal("-1E10");

        BigDecimal firstOrder = x.add(y).add(z);
        System.out.println("(x + y) + z = " + firstOrder);

        BigDecimal secondOrder = x.add(y.add(z));
        System.out.println("x + (y + z) = " + secondOrder);
    }
}

```

In this example, `BigDecimal` provides arbitrary precision, so  $(x + y) + z$  equals  $x + (y + z)$ , demonstrating the associative property of addition.

Do note that `BigDecimal` operations can be slower than primitive floating-point operations, due to the overhead of object creation and the more complex algorithms used for high-precision arithmetic. Hence, `BigDecimal` should be used when the precision requirement outweighs the need for speed.

**You:** the same Java example but using `float`?

**ChatGPT:** Sure! Here's the example in Java using `float`:

```
public class Main {
    public static void main(String[] args) {
        float x = 1.0e10f;
        float y = 1.0f;
        float z = -1.0e10f;

        float firstOrder = (x + y) + z;
        System.out.println("(x + y) + z = " + firstOrder);

        float secondOrder = x + (y + z);
        System.out.println("x + (y + z) = " + secondOrder);
    }
}
```

In this example, just like in the C example,  $(x + y) + z$  is not equal to  $x + (y + z)$  due to floating point precision errors. Java's `float` has a precision of about 7 decimal digits, so it is not able to represent the result of  $x + y$  accurately because  $y$  is significantly smaller than  $x$ . Therefore, the result of  $(x + y) + z$  is 1.0 due to the loss of precision in  $x + y$ . On the other hand,  $y + z$  equals  $-1.0e10$  exactly, and then adding  $x$  results in 0.0.