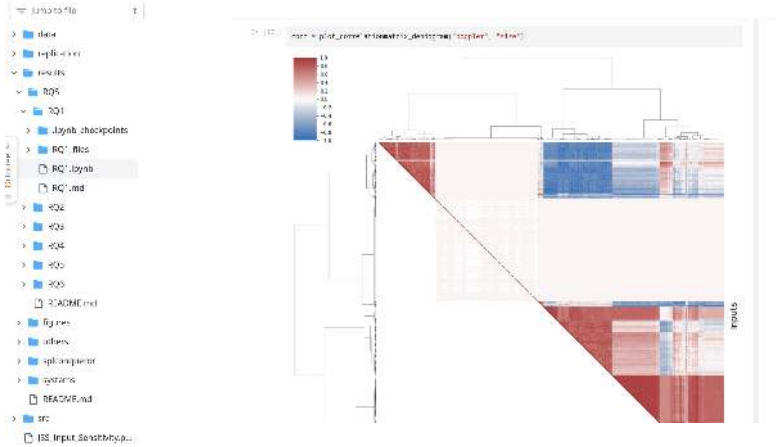
Tackling Deep Software Variability Together

Mathieu Acher @acherm

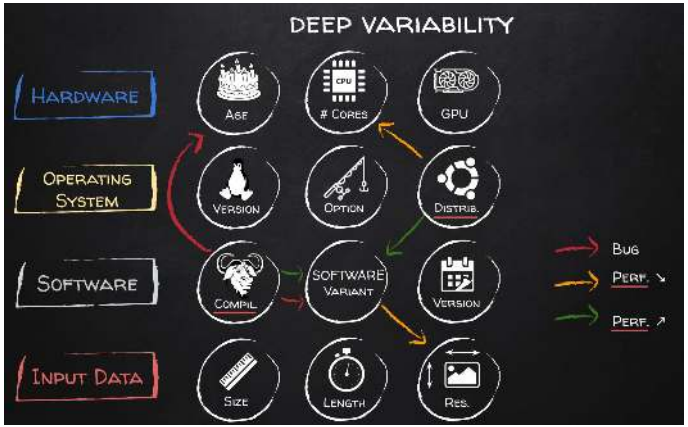
Special thanks to Luc Lesoil, Jean-Marc Jézéquel, Juliana Alves Pereira, and Paul Temple

DiverSE
Diversity-Centric Software Engineering

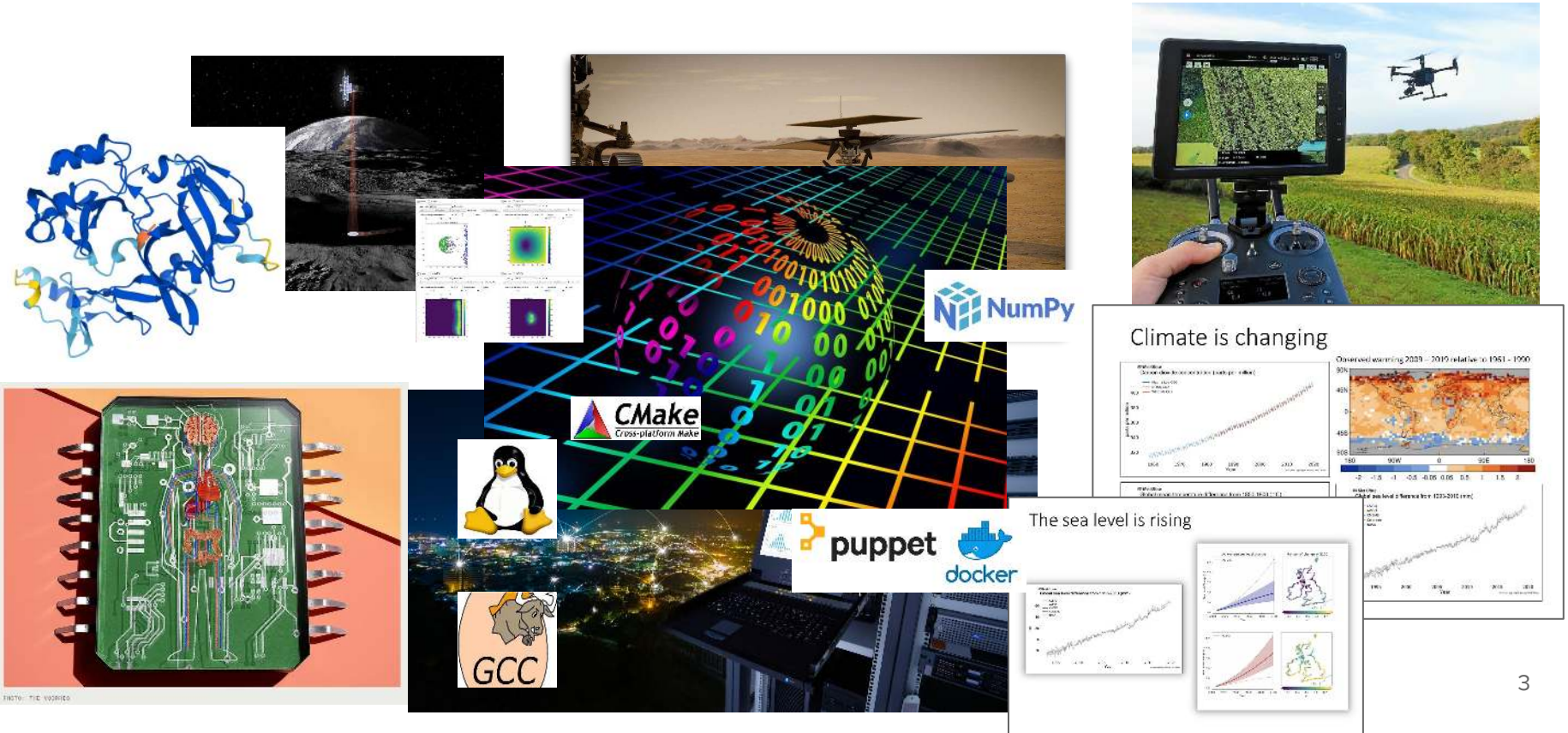
institut universitaire de France



Special thanks to Luc Lesoil, Jean-Marc Jézéquel, Juliana Alves Pereira, and Paul Temple

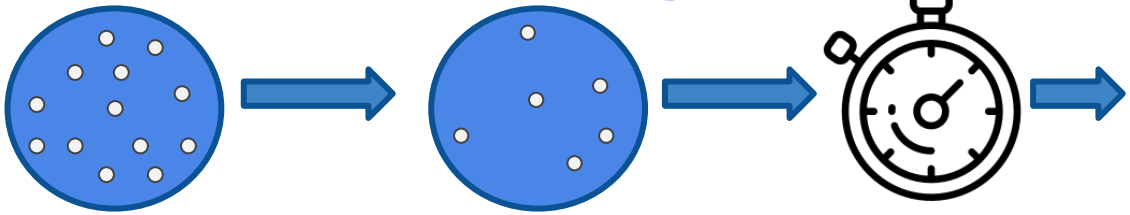


SOFTWARE VARIANTS ARE EATING THE WORLD AND SCIENCE




build: OK
8.16 seconds
security:
high
build: failure
build: OK
2.16 seconds
security: low

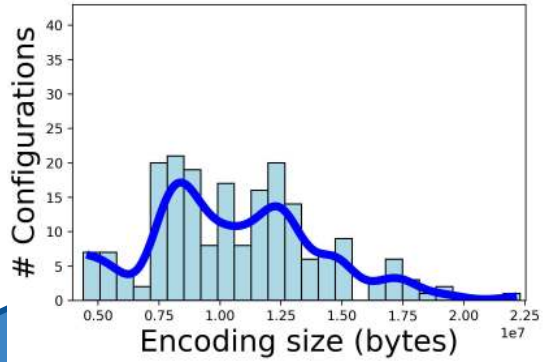
Sampling, Measuring, Learning Software Configuration Spaces



Whole Population of **Configurations**

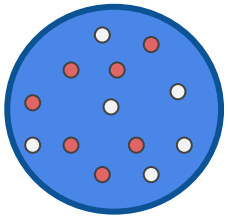
Training Sample

Performance Measurements

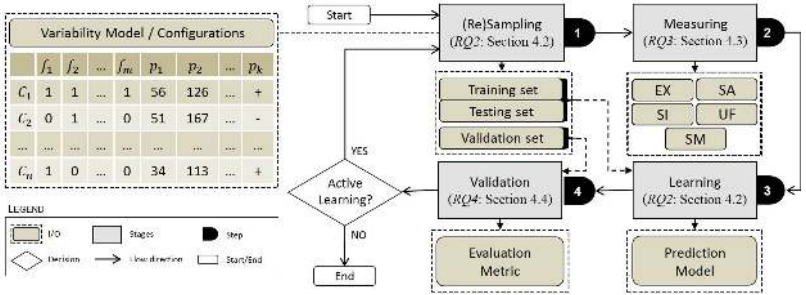


Prediction Model

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



Performance Prediction



J. Alves Pereira, H. Martin, M. Acher, J.-M. Jézéquel, G. Botterweck and A. Ventresque "Learning Software Configuration Spaces: A Systematic Literature Review" JSS, 2021

Learning variability can be easy...

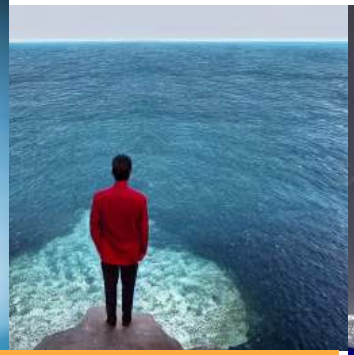
State-of-the-art (SOTA) results: with the right algorithm and/or sampling strategy, fairly accurate and interpretable models for many systems and qualitative/quantitative properties can be derived

Still, why learning-based, SOTA techniques are not fully adopted and integrated to software-intensive projects?

- Academic/industry gap?
- Too costly?
- Hard to automatically observe/quantify variants?
- ….

References	Name	Domain	Non-Functional Properties
[61, 87]	Wget	Data transfer	memory footprint, code complexity
[106]	Actian Vector	Database system	runtime
[41]	Apache Cassandra	Database system	latency
[27, 29, 42, 48, 61, 62, 66, 80, 84–89, 98, 118, 119]	Berkeley DB	Database system	I/O time, memory footprint, performance, response time, code complexity, maintainability, binary size
[89]	FAME-DBMS	Database system	maintainability, binary size, performance
[93, 106, 117, 120]	MySQL	Database system	defects, throughput, latency
[106]	Postgres	Database system	throughput, latency
[86–88]	Prevayler	Database system	memory footprint, performance
[27, 29, 39, 47, 61, 62, 80, 85–87, 87, 88, 98, 105]	SQLite	Database system	memory footprint, performance, response time, code complexity, runtime
[15]	StockOnline	Database system	response time
[10]	Kafka	Distributed systems	throughput
[25]	DNN	DNNs algorithms	accuracy of predictions
[3]	Curriculum vitae	Document	number of pages
[3]	Paper	Document	number of pages
[20]	RUBiS	E-commerce application	response time
[91]	EMAIL	E-mail client	time
[47]	MBED TLS	Encryption library	response time
[110]	SAP ERP	Enterprise Application	response time
[63]	noc-CM-log	FPGA	CPU power consumption, runtime
[63]	sort-256	FPGA	area, throughput
[23]	E-Health System	Health	response time
[29, 42, 84, 98]	*HIPA*^{cc}	Image processing	response time
[17]	Disparity SPL	Image processing	energy consumption
[86–88]	PKJab	Instant messenger	memory footprint, performance
[36]	IBM ILOG CPLEX	Integer solver	runtime
[110]	SPECjjbb2005	Java Server	response time, throughput
[101]	WEKA	Learning algorithm	accuracy of predictions
[18]	SVD	Linear algebra	execution time and accuracy
[63]	Trimesh	Mesh solver	iterations, response time
[91]	MBENCH	Micro benchmark	time
[73, 116]	ACE+TAO system	Middleware software	defects
[86–88]	SensorNetwork	Network simulator	memory footprint, performance
[109]	Simonstrator	Network simulator	latency
[121]	NoC	Network-based system	energy and runtime
[18]	Helmholtz 3D	Numerical analysis	execution time and accuracy
[18]	Poisson 2D	Numerical analysis	execution time and accuracy
[86–88, 92]	Linux kernel	Operating system	memory footprint, performance
[40]	DNN	Optimization algorithm	response time
[55]	Art system	Paint	user likeability
[26]	Multigrid system	Equations solving	time of each interaction
[41]	CoBot System	Robotic system	CPU usage
[42, 84, 98]	JavaGC	Runtime environment	response time
[96]	Robot	Runtime environment	energy consumption and execution time
[61–63, 66, 80, 84, 85, 87, 98, 105, 118, 119]			time, Peak Signal to Noise Ratio, response time, code complexity, video quality, performance
[98]	OpenCV	Video tracking	performance
[77]	C60 and MX300	Virtual environment	defects
[6]	Amazon EC2	Web coud service	performance
[27, 29, 48, 61, 62, 66, 80, 84, 85, 98, 117, 120]	Apache	Web server	response rate, response time, workload, defects, throughput
[67]	Stock Brokerage	Web system	throughput, response time
[93]	vsftpd	FTP daemon	defects
[93]	ngIRCd	IRC daemon	defects

Learning variability can be easy…



… but we're not looking deeply.

Prediction models and configuration knowledge may not generalize (ie misleading/inaccurate/pointless for developers and users) if the software evolves, is compiled differently, a different hardware is used, input data fed differs, etc.

software application variability

input data variability

build/compiler variability

version variability

operating system variability

hardware variability

deep software variability

hardware variability

15,000+ options

thousands of compiler flags and compile-time options

dozens of preferences

100+ command-line parameters

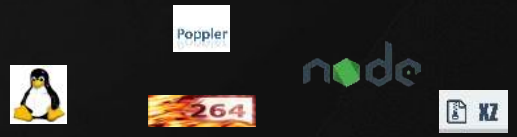
1000+ feature toggles

DEEP VARIABILITY

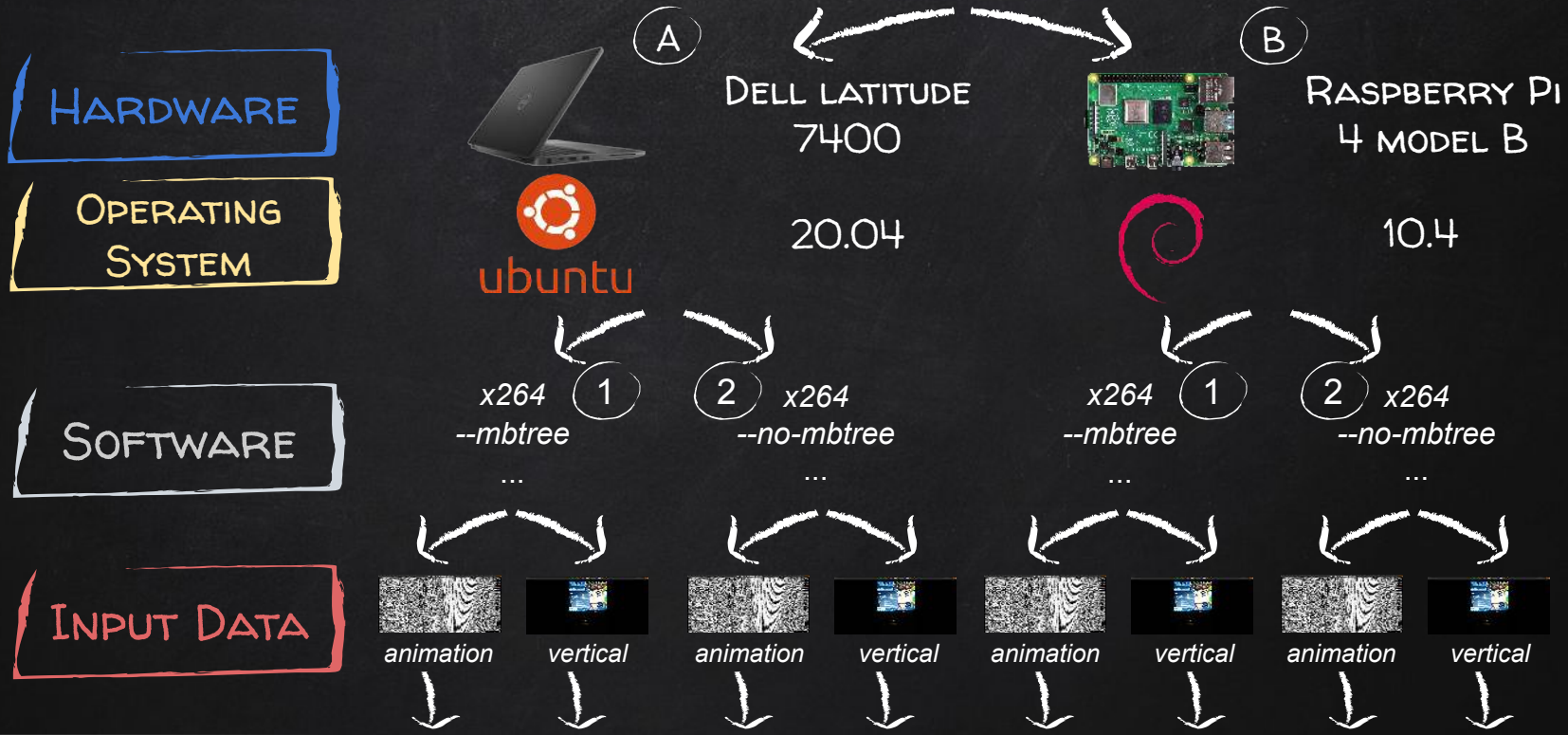
Sometimes, variability is consistent/stable across layers and knowledge transfer is immediate.

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

- BUG
- PERF. ↘
- PERF. ↗



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)

Hardware

A — Dell Latitude 7400

B — Raspberry Pi 4 model B

Operating System

ubuntu 20.04

10.4

Software

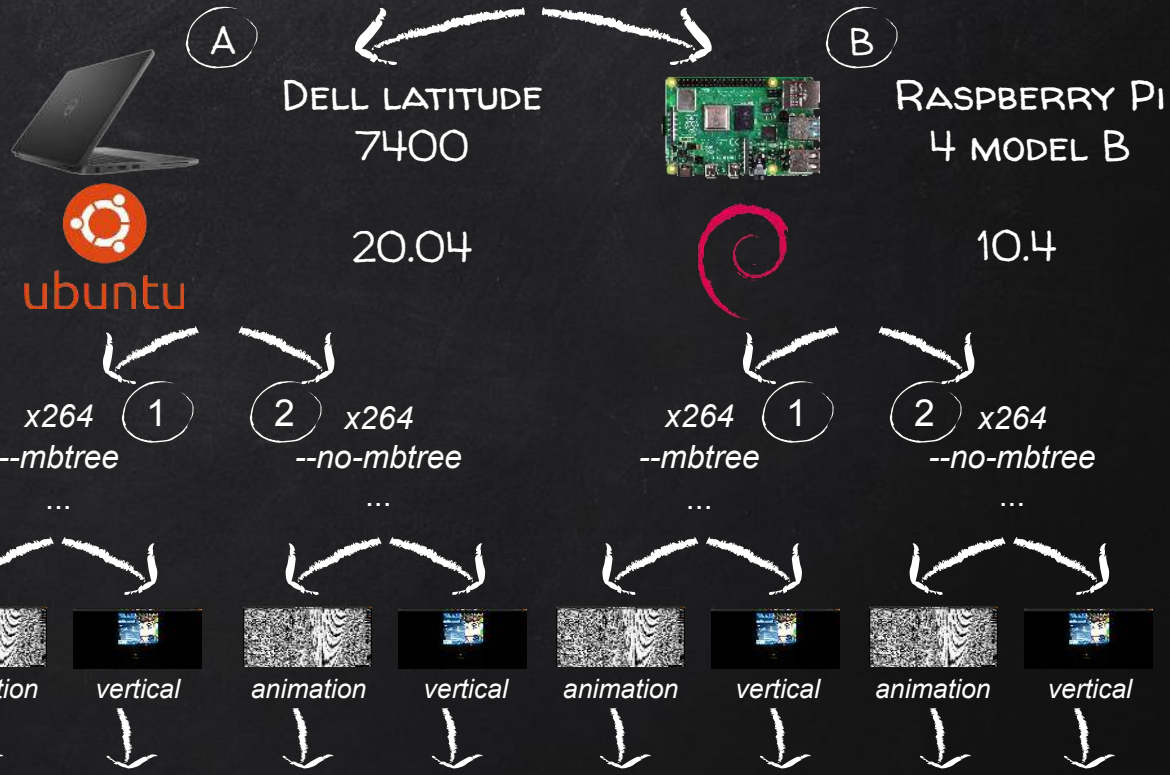
	A (ubuntu 20.04)		B (10.4)	
1	x264 --mbtree ...		x264 --mbtree ...	
2	x264 --no-mbtree ...		x264 --no-mbtree ...	

Input Data

	A (ubuntu 20.04)		B (10.4)	
	animation	vertical	animation	vertical

	A (ubuntu 20.04)				B (10.4)			
	1		2		1		2	
	animation	vertical	animation	vertical	animation	vertical	animation	vertical
Duration (s)	6	22	6	25	73	351	72	359
Size (MB)	33	28	21	34	33	28	21	34

Real World Example (x264)



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

DEEP VARIABILITY

The "best"/default software variant might be a bad one.

Influential software options and their interactions vary.

Performance prediction models and **variability knowledge may not generalize**

BUG

PERF. ↘

PERF. ↗

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)
Analysis time	**Encoding time**	**Query time**	**Execution time**
14 options	16 options	14 options	50 options
16,384 configurations	4,000 configurations	1,000 configurations	71,267 configurations
SAT problems	Video quality/size	DB Queries	10 Demo programs
3 hardware	2 hardware	2 hardware	
2 versions	3 versions	2 versions	

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

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

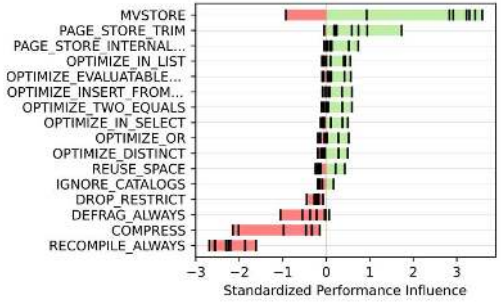
Insight. The strength of the influence of configuration options is typically preserved across environments.

Insight. A large percentage of configurations are typically invalid in both source and target environments.

“Analyzing the Impact of **Workloads** on Modeling the Performance of Configurable Software Systems”
Stefan Mühlbauer et al. ICSE 2023

Summary (RQ_1): Varying the workload causes a substantial amount of variation among performance distributions. Across workloads, we observed *mostly linear* (for six of the nine subject systems), but to a large extent, also *non-monotonous* relationships (for three of the nine subject systems).

Summary (RQ_2): Workloads can affect performance influences of configuration options in various ways (e.g., conditioning influence, introducing variance, having outliers). We can correlate some variation of performance influences with workload characteristics, yet identifying relevant workload characteristics is highly domain-specific and cannot be considered trivial.



(c) Distribution of configuration options' performance influences under different workloads for H2.

Table IV: Common top five influential configuration options among pairs of workloads.

System	Workload 1	Workload 2	# Common
JUMP3R	helix.wav	sweep.wav	2
KANZI	vmlinux	fannie_mae_500k	1
DCONVERT	jpeg-small	svg-large	2
H2	tpcc-2	tpcc-8	3
BATIK	village	cubus	4
XZ	deepfield	silesia	4
LRZIP	artificl	uiq-32-bin	3
X264	sd_crew_cif_short	sd_city_4cif_short	4
Z3	QF_NRA_hong_9	QF_BV_bench_935	3

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?

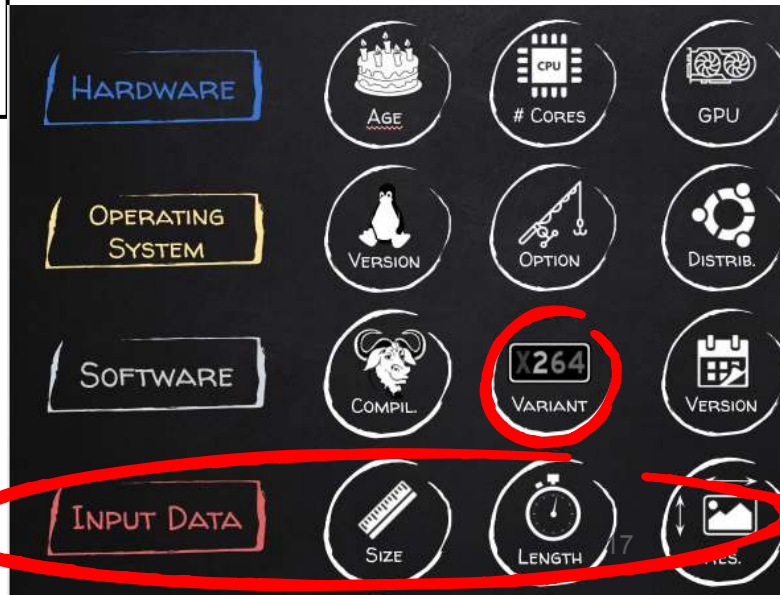
Are influential options always influential?

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.

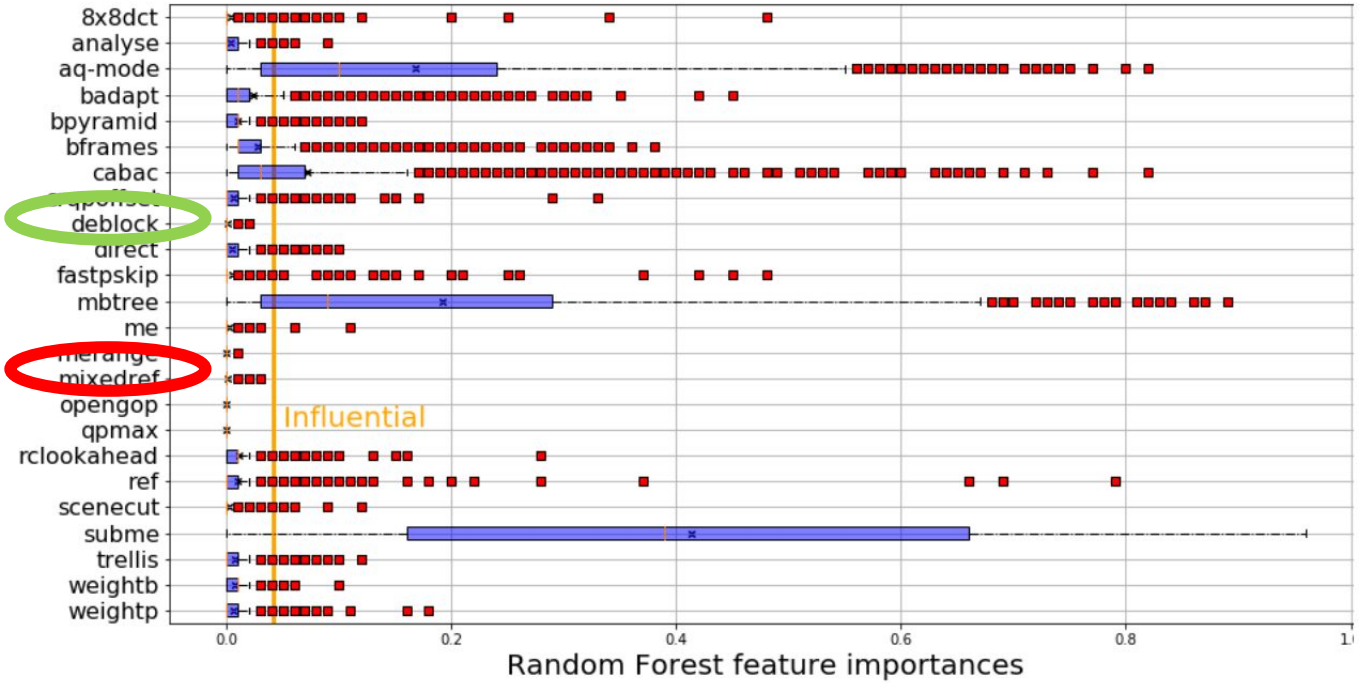
Do x264 software performances stay consistent across inputs?

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? (P = bitrate)



L. Lesoil, M. Acher, A. Blouin and J.-M. Jézéquel “Input sensitivity on the performance of configurable systems an empirical study” JSS 2023

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

eg effectiveness of sampling strategies (random, 2-wise, etc.) is input specific (see also Pereira et al. ICPE 2020)

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

Stefan Mühlbauer, Florian Sattler, Christian Kaltenecker, Johannes Dorn, Sven Apel, Norbert Siegmund “Analyzing the Impact of Workloads on Modeling the Performance of Configurable Software Systems” ICSE 2023

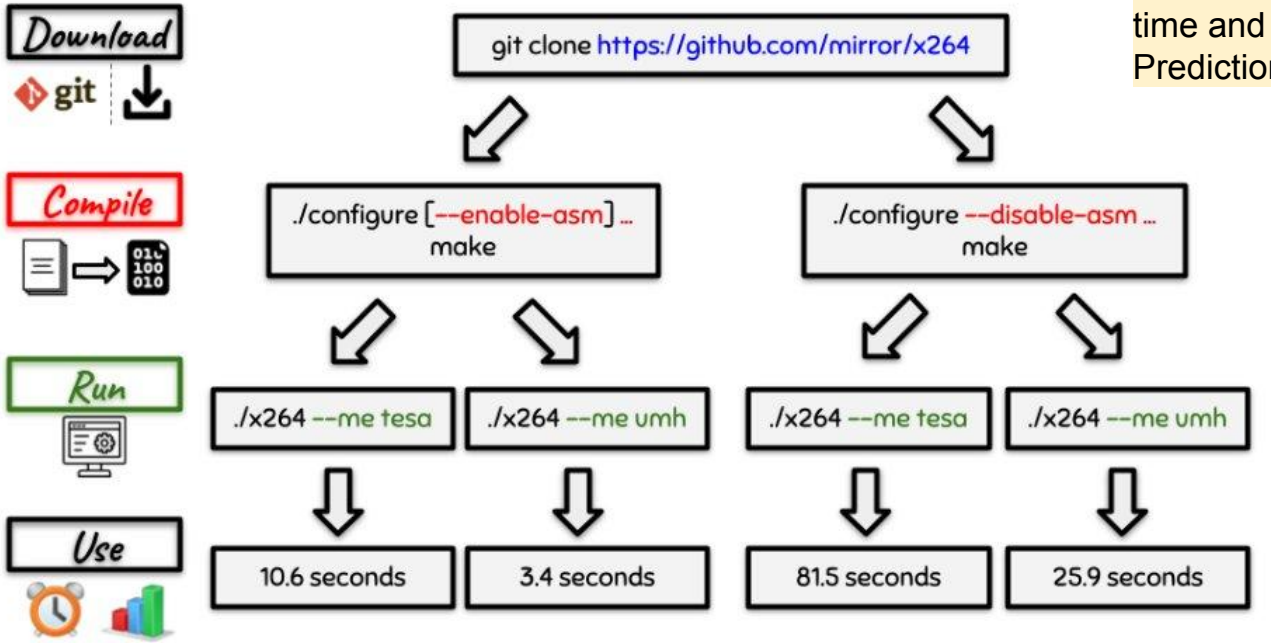
Insight: Workload sensitivity challenges the robustness and generalizability of single-workload performance models, yet it is neglected in state-of-the-art approaches. Worse, robust techniques using only rankings or relative importance of options are inapplicable for certain workload variations.

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 “Input sensitivity on the performance of configurable systems an empirical study” JSS 2023

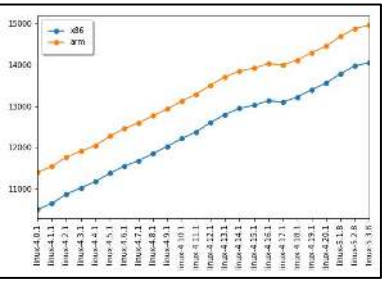
Is there an interplay between compile-time and runtime options?

L. Lesoil, M. Acher, X. Tërnava, 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.

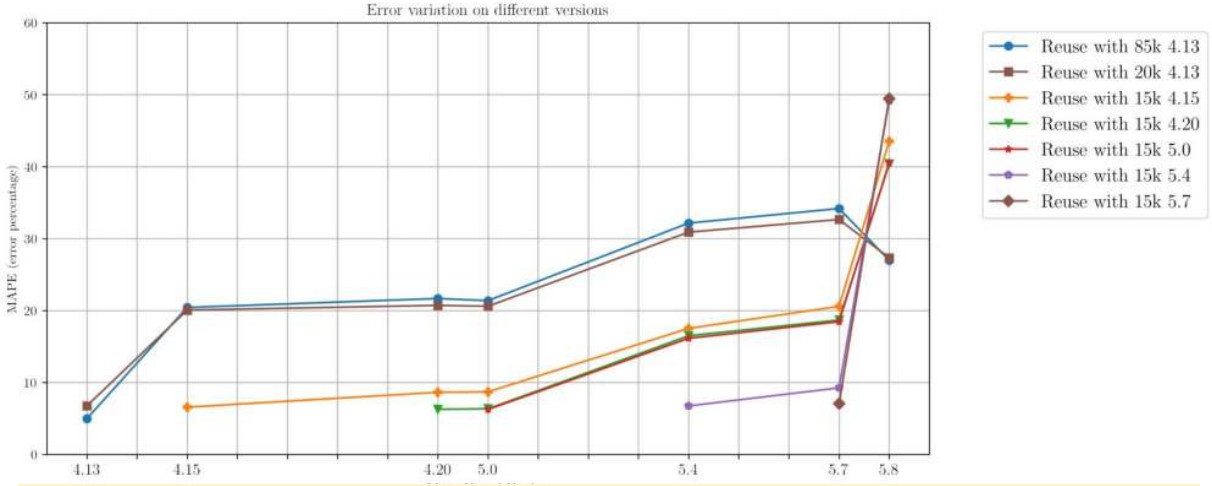
Figure 1: Cross-layer variability of x264

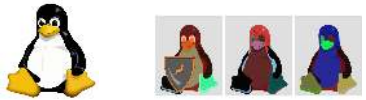


Version	Release Date	LOC	Files	Examples	Seconds/config	Options	Features	Deleted features	New features	ΔCommits	Files changes
4.13	2017/09/03	16,616,534	60,530	92,562	not available	12,776	9,468	-	-	-	-
4.15	2018/01/28	17,073,368	62,249	39,391	not available	12,998	9,425	342	299	31,052	934,628
4.20	2018/12/23	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

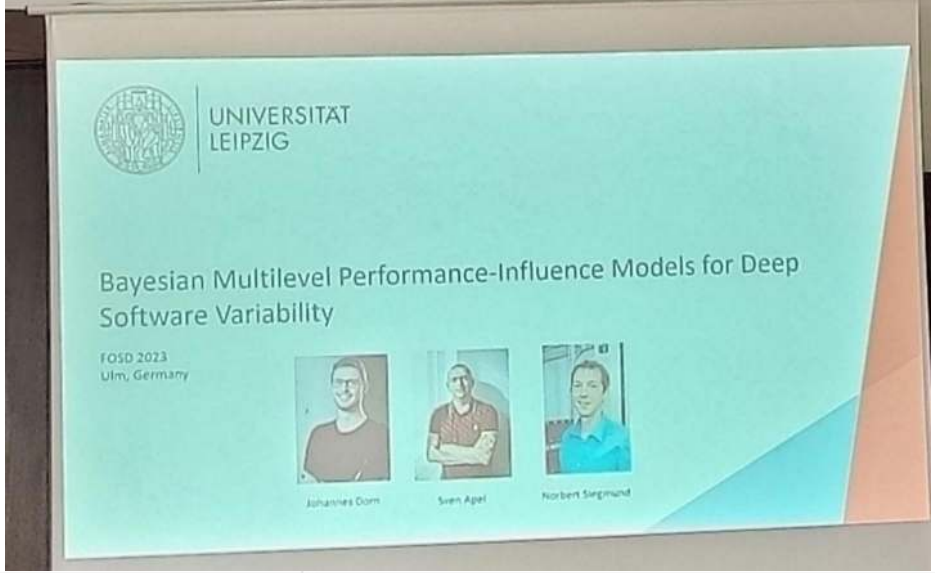
Table I: Dataset properties for each version. The number of deleted/new features, delta commits, files changes are w.r.t. 4.13.

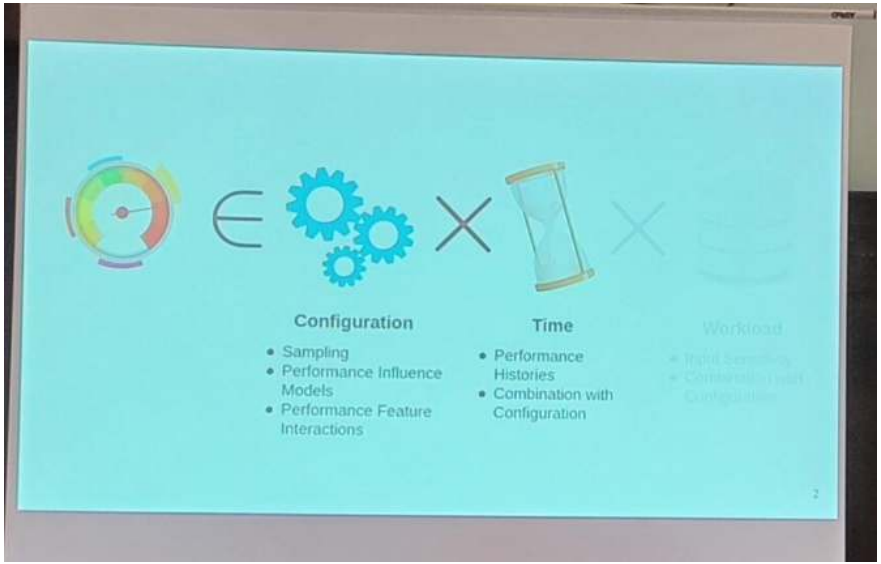
4.13 version (sep 2017): 6%. What about evolution? Can we reuse the 4.13 Linux prediction model? No, accuracy quickly decreases: **4.15 (5 months after): 20%; 5.7 (3 years after): 35%**





H. Martin, M. Acher, J. A. Pereira, L. Lesoil, J. Jézéquel and D. E. Khelladi, “Transfer learning across variants and versions: The case of linux kernel size” Transactions on Software Engineering (TSE), 2021



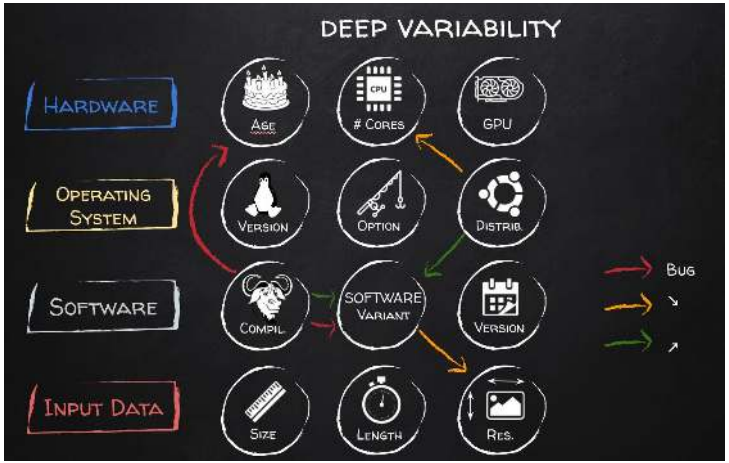


Stefan Mühlbauer	Group Sampling for Learning Large Configuration Spaces

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?

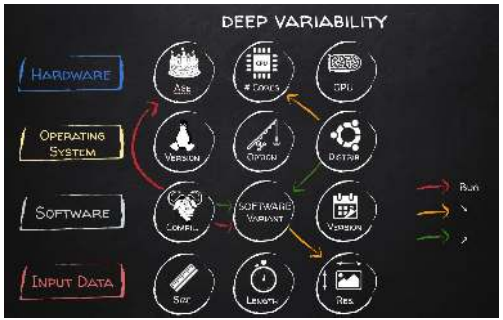
Expected outcomes:

- 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)

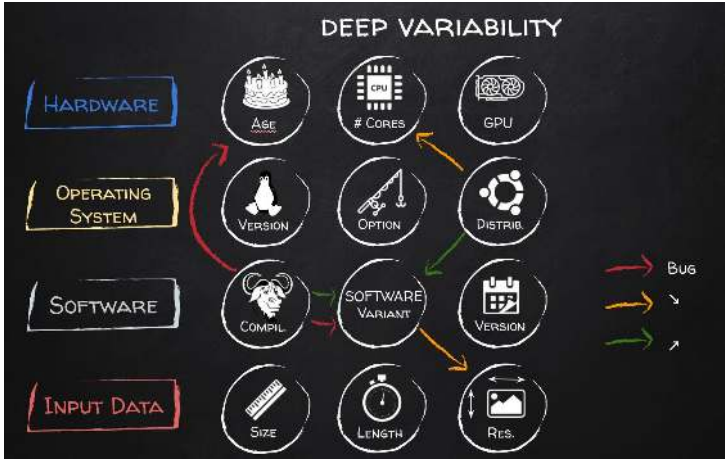
Reducing the cost of exploring the variability spaces

- learning
  - many algorithms/techniques with tradeoffs interpretability/accuracy
  - transfer learning (instead of learning from scratch) Jamshidi et al. ASE'17, ASE'18, Martin et al. TSE'21
- sampling strategies
  - uniform random sampling? t-wise? distance-based? …
  - sample of hardware? input data?
- incremental build of configurations Randrianaina et al. ICSE'22
- white-box approaches Velez et al. ICSE'21, Weber et al. ICSE'21
- …



What can we do? (#3 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

Open, reproducible science for deep variability

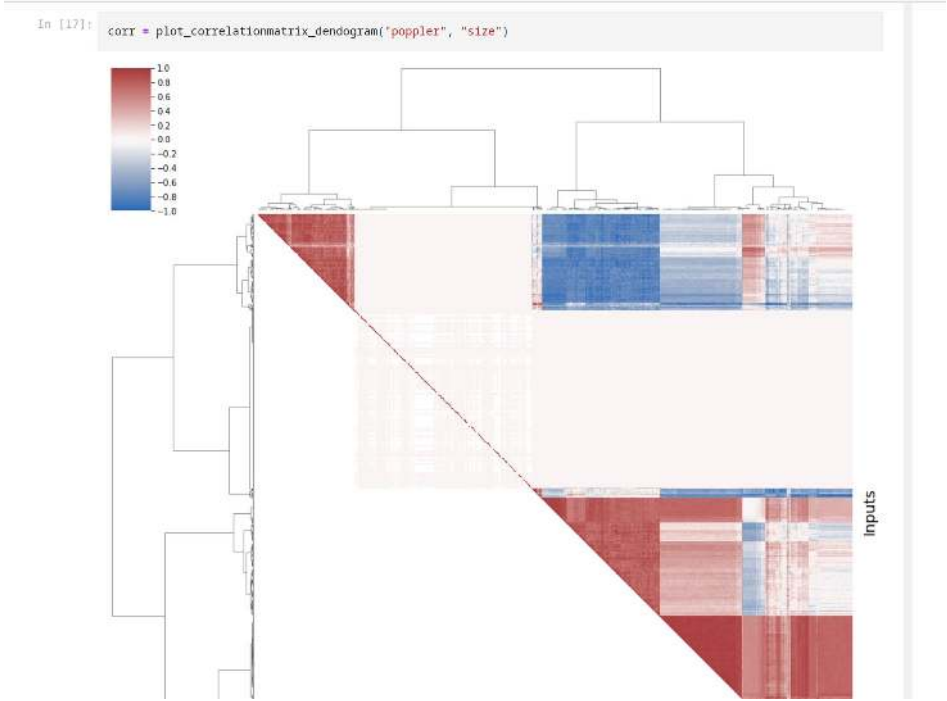
deep.variability.io?

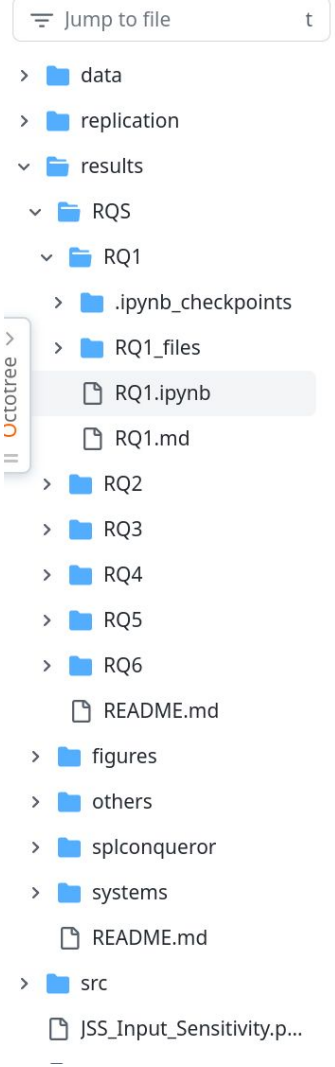
A collaborative resource and place:

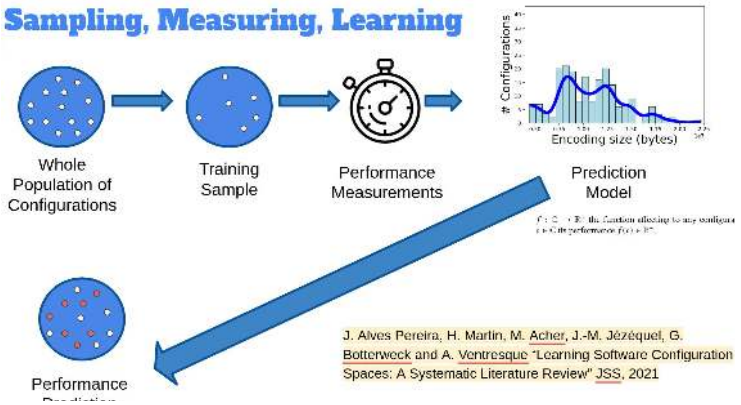
- Evidence of Deep Software Variability
- Continuous survey of papers/articles about deep software variability
- Case studies and configuration knowledge associated to software projects
- Datasets (eg measurements of performance in certain conditions)
- Reproducible scripts (eg for building prediction models)
- Description and results of Innovative solutions (eg transfer learning)
  - content based on already published papers
  - beyond PDFs

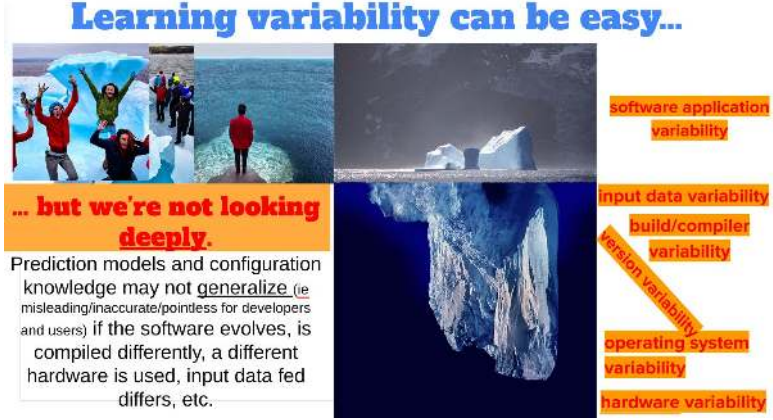
Ongoing work (“live” book with jupyter)… Looking for contributions/ideas/suggestions!

eg https://github.com/llesoil/input_sensitivity



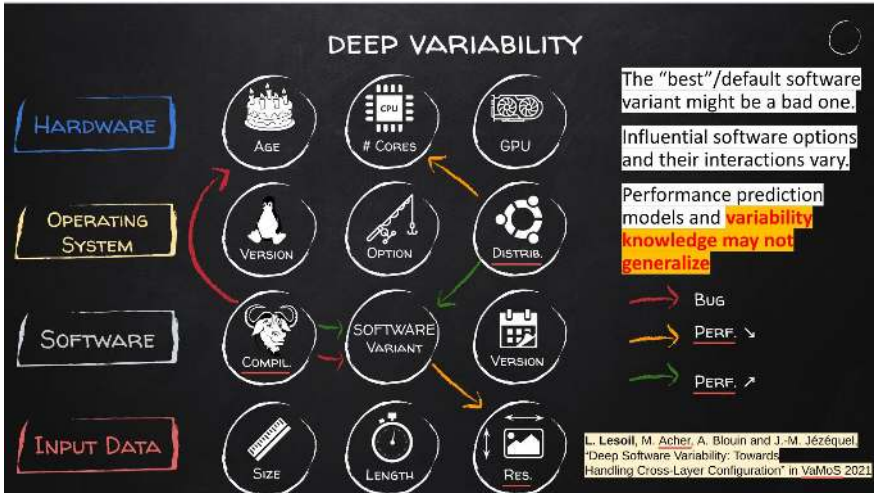






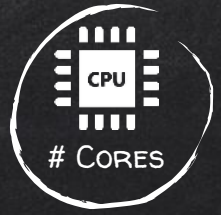
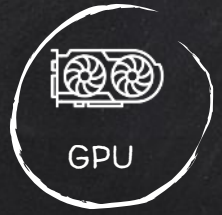
Tackling Deep Software Variability Together

Mathieu Acher @acherm



BACKUP SLIDES

(BEYOND x264) EMPIRICAL AND FUNDAMENTAL QUESTION

HARDWARE	 AGE	 # CORES	 GPU
OPERATING SYSTEM	 VERSION	 OPTION	 DISTRIB.
SOFTWARE	 COMPIL.	VARIANT	 VERSION
INPUT DATA	 SIZE	 LENGTH	 RES.

HOW DOES DEEP SOFTWARE VARIABILITY MANIFEST IN THE WILD?

SOFTWARE SCIENTISTS SHOULD OBSERVE THE JUNGLE/ GALAXY!

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

- **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



Operating System



Software

0.152.2854

0.155.2917

Input Data



Problem

Combinatorial explosion and cost

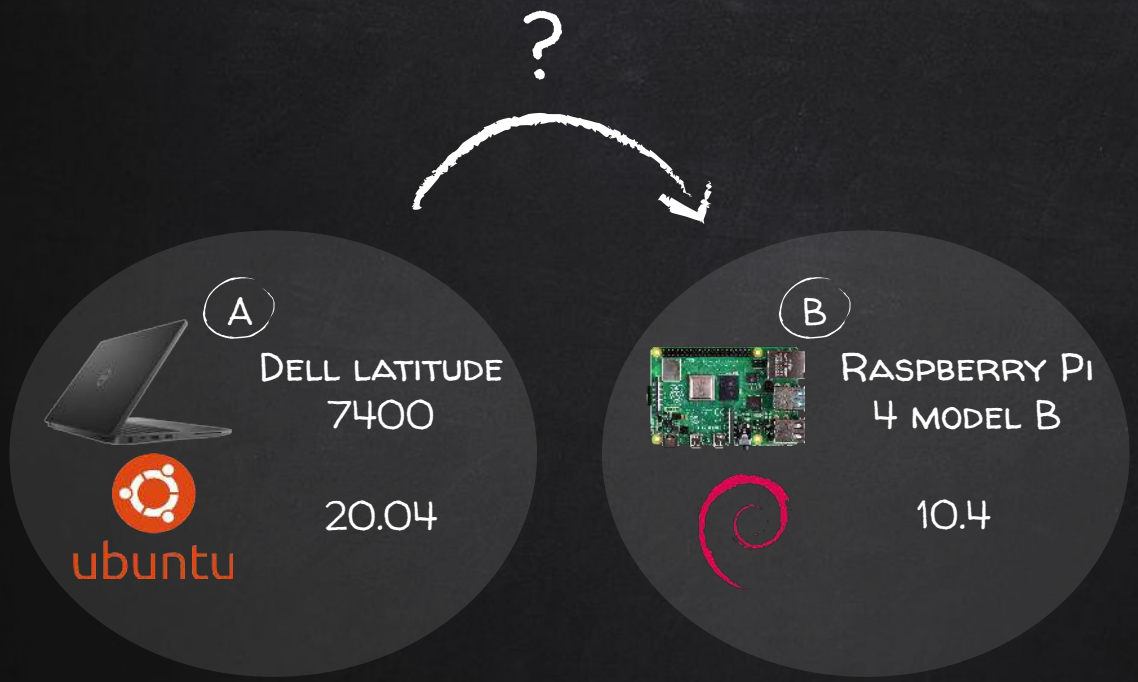
Challenge

Build a representative, cheap set of environments

Opportunity

dimensionality reduction

Transfer performances across environments



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

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
```

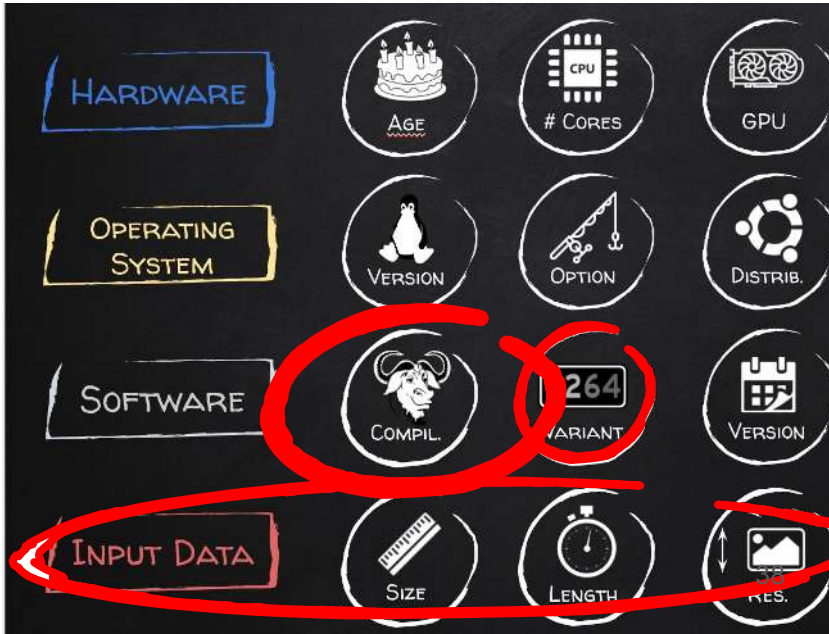
```
  --disable-thread         disable multithreaded encoding
  --disable-win32thread    disable win32threads (windows only)
  --disable-interlaced     disable interlaced encoding support
  --bit-depth=BIT_DEPTH    set output bit depth (8, 10, all) [all]
  --chroma-format=FORMAT   output chroma format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



Key results (for x264)

L. Lesoil, M. Acher, X. Tërnava, A. Blouin and J.-M. Jézéquel “The Interplay of Compile-time and Run-time Options for Performance Prediction” in SPLC ’21

***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).

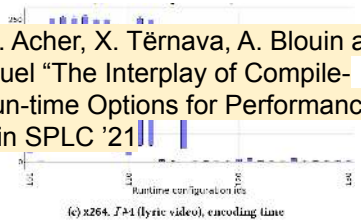
Reuse of configuration knowledge: $f_1 = \beta \times f_2 + \alpha$

- Linear transformation among distributions
- Users can also trust the documentation of run-time options,

consistent whatever the compile-time configuration is.

Key results (Poppler)

L. Lesoil, M. Acher, X. Tërnava, A. Blouin and J.-M. Jézéquel “The Interplay of Compile-time and Run-time Options for Performance Prediction” in SPLC ’21



(e) x264, JM1 (lyric video), encoding time

First good new: *Worth t*

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

Second good *dev*: all th input videos, the worst correlation is creat scaled the distribution, they do not change the r kings era with the run me options).

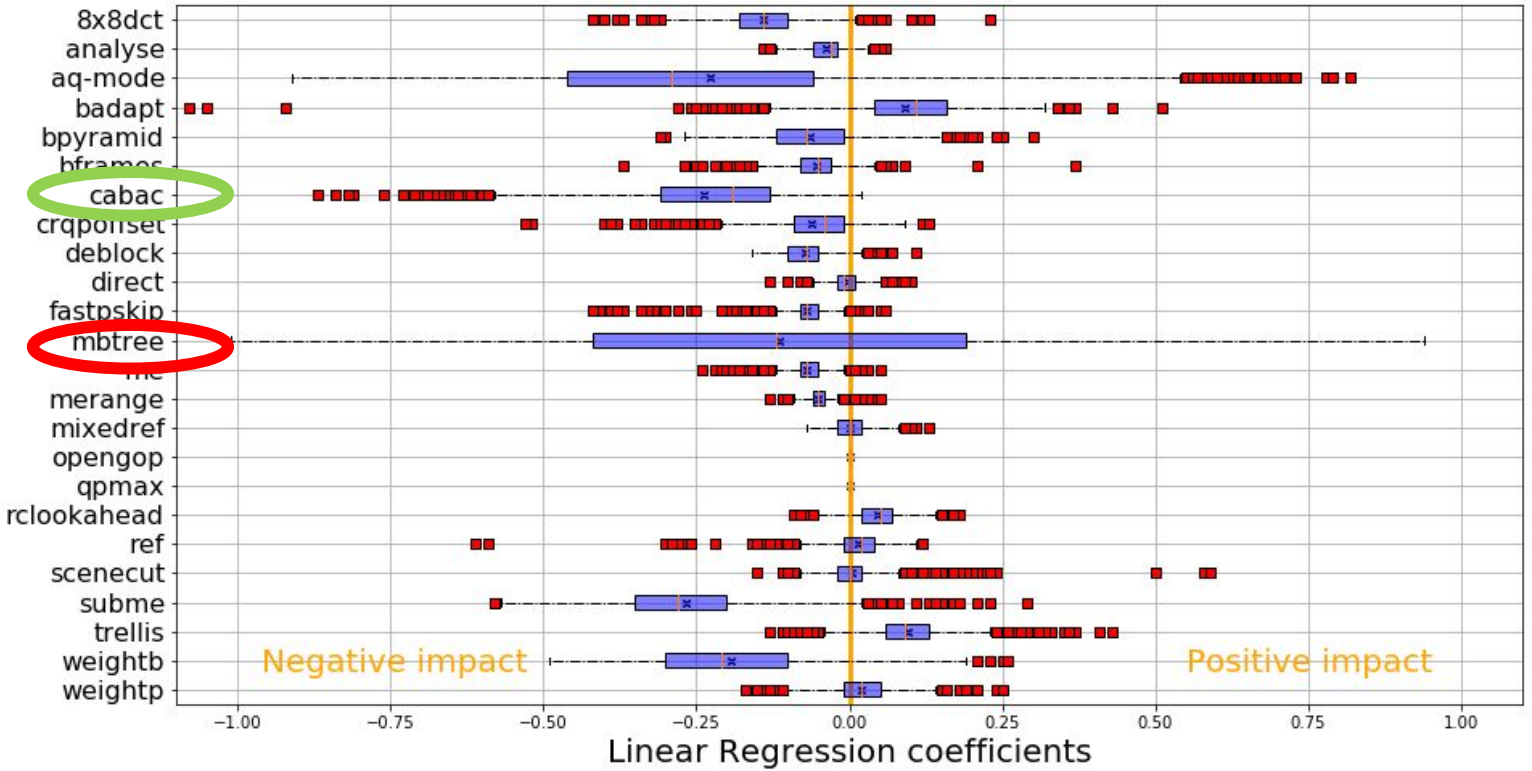
It has three pr tical implications:

transfer lear g o s l s do o ply a ear
Users can ust e tio of run- e ptions,
e configurat n is.
st compile-time configuration among all the possible ones allows
nfiguration at run time. We can remove aw
a default compile-time configuration, use

rie r for

performances!

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



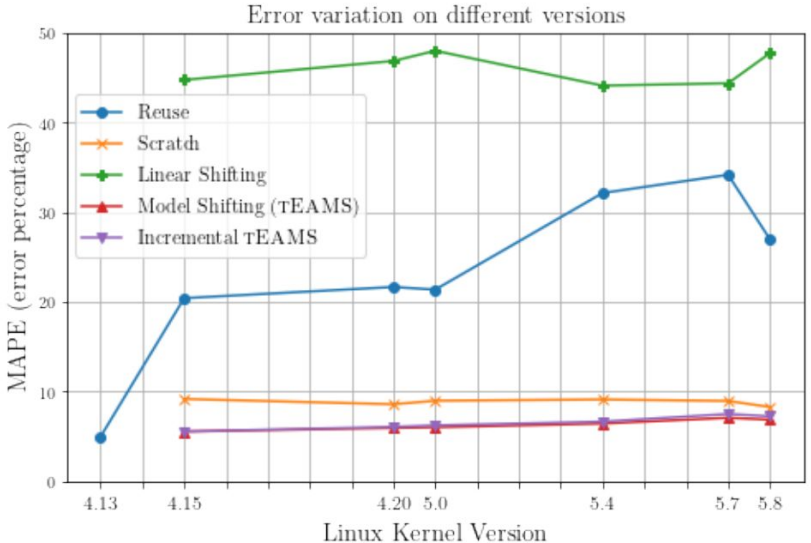
H. Martin, M. Acher, J. A. Pereira, L. Lesoil, J. Jézéquel and D. E. Khelladi, “Transfer learning across variants and versions: The case of linux kernel size” Transactions on Software Engineering (TSE), 2021

- 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…

H. Martin, M. Acher, J. A. Pereira, L. Lesoil, J. Jézéquel and D. E. Khelladi, “Transfer learning across variants and versions: The case of linux kernel size” Transactions on Software Engineering (TSE), 2021

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



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

Deep Reinforcement Learning that Matters

Peter Henderson[1]*, Riashat Islam[1,2]*, Philip Bachman[2]
Joelle Pineau[1], Doina Precup[1], David Meger[1]

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?

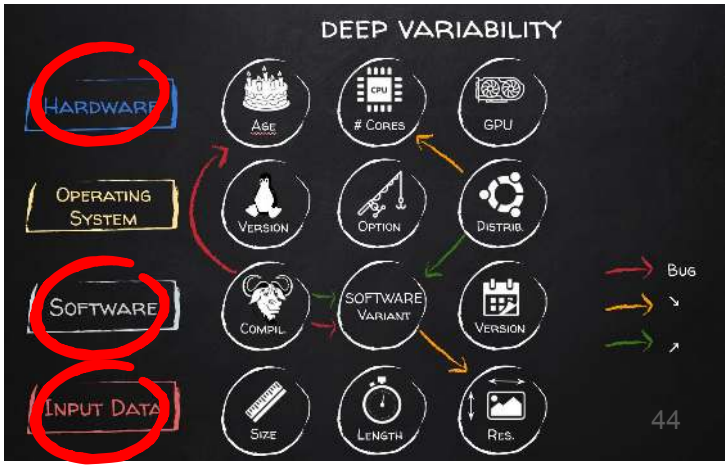
Are commonly used baseline **implementations** comparable?

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

David Picard DAVID.PICARD@ENPC.FR
LIGM, École des Ponts, 77455 Marnes la vallée, France

Abstract

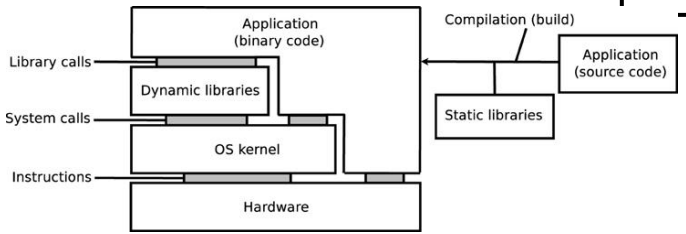
In this paper I investigate the effect of random seed selection on the accuracy when using popular deep learning architectures for computer vision. I scan a large amount of seeds (up to 10^4) on CIFAR 10 and I also scan fewer seeds on Imagenet using pre-trained models to investigate large scale datasets. 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.



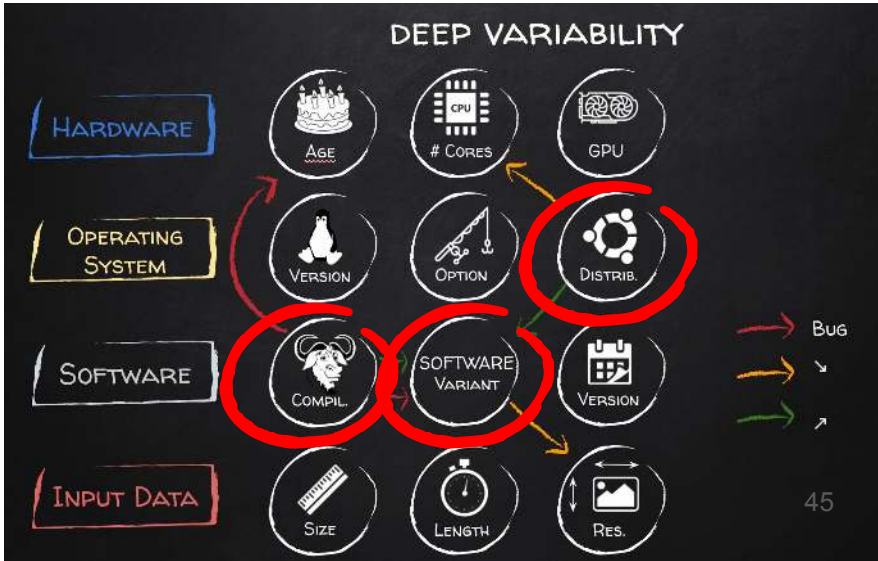
“Neuroimaging pipelines are known to generate different results depending on the computing platform where they are compiled and executed.”

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**	Freesurfer 5.3.0, build **1** and **2**
	FSL 5.0.6, build **1**	FSL 5.0.6, build **1** and **2**
	CIVET 1.1.12-UCSF, build **1**	CIVET 1.1.12-UCSF, build **1**
Interpreters	`Python` 2.4.3, `bash` 3.2.25,	`Python` 2.7.5, `bash` 4.2.47,
	`Perl` 5.8.8, `tcsh` 6.14.00	`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)



Deep Questions?