The Interaction between Inputs and Configurations fed to Software Systems: an Empirical Study

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

Noname manuscript No.
(will be inserted by the editor)

The Interaction between Inputs and Configurations fed to Software Systems: an Empirical Study

Luc Lesoil, Mathieu Acher, Arnaud Blouin, Jean-Marc Jézéquel.
Univ Rennes, INSA Rennes, IRISA, Inria. France.
firstname.lastname@irisa.fr

the date of receipt and acceptance should be inserted later

Abstract Widely used software systems such as video encoders are by necessity highly configurable, with hundreds or even thousands of options to choose from. Their users often have a hard time finding suitable values for these options (*i.e.*, finding a proper configuration of the software system) to meet their goals for the tasks at hand, *e.g.*, compress a video down to a certain size. One dimension of the problem is of course that performance depends on the input data: *e.g.*, a video as input to an encoder like *x264* or a file system fed to a tool like *xz*. To achieve good performance, users should therefore take into account both dimensions of (1) software variability and (2) input data. In this problem-statement paper, we conduct a large study over 8 configurable systems that quantifies the existing interactions between input data and configurations of software systems. The results exhibit that (1) **inputs fed to software systems interact with their configuration options in non monotonous ways**, significantly impacting their performance properties (2) tuning a software system for its input data makes it possible to multiply its performance by up to ten (3) input variability can jeopardize the relevance of performance predictive models for a field deployment.

Keywords Input Sensitivity, Software variability, Performance prediction

1 Introduction

Widely used software systems are by necessity highly configurable, with hundreds or even thousands of options to choose from. For example, a tool like *xz* offers multiple options such as –threads or –format for compressing a file. The same applies to *Linux* kernels or video encoders such as *x264*: they all provide configuration options through compilation options, feature toggles, and command-line parameters. Software engineers often have a hard time finding suitable values for those options (*i.e.*, finding a proper configuration of the

Address(es) of author(s) should be given

software system) to meet their goals for the tasks at hand, *e.g.*, compile a high-performance binary or compress a video down to a certain size while keeping its perceived quality. Since the number of possible configurations grows exponentially with the number of options, even experts may end up recommending sub-optimal configurations for such complex software [38].

However, there exist cases where inputs (*e.g.*, files fed to an archiver like *xz* or SAT formulae provided as input to a solver like *lingeling*) can also impact software variability [52, 98]. The *x264* encoder typifies this problem. For example, Kate, an engineer working for a VOD company, wants *x264* to compress input videos to the smallest possible size. As illustrated in Figure 1, she executes *x264* with two configurations C (with options –no-mbtree –ref 1) and C' (with options –no-cabac –ref 16) on the input video I1 and states that C is more appropriate than C' in this case. But when trying it on a second input video I2, she draws opposite conclusions; for I2, C' leads to a smaller output size than C. Now, Kate wonders what configuration to choose for other inputs, C or C'? More generally, do configuration options have the same effect on the output size despite a different input? Do options interact in the same way no matter the inputs? These are crucial practical issues: the diversity of existing inputs can alter her knowledge of *x264*'s variability. If it does, Kate would have to configure *x264* as many times as there are inputs, making her work really tedious and difficult to automate for a field deployment.

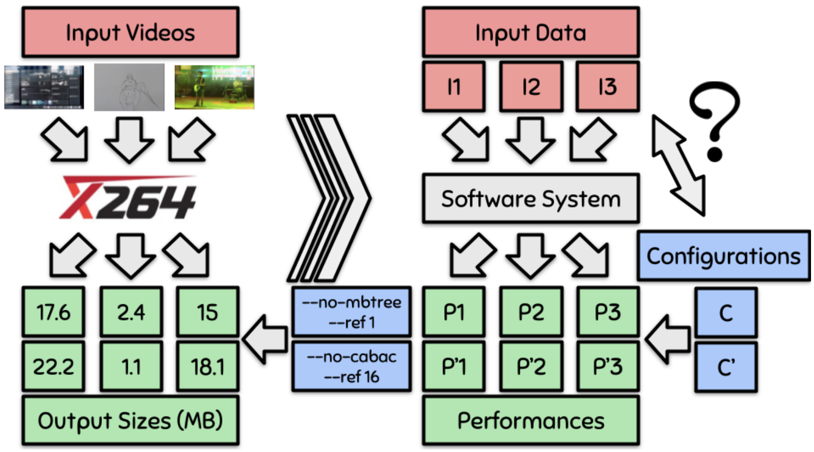


Fig. 1: This paper explores and quantifies how inputs fed to software systems interact with their configuration options.

In this work we conduct, to our best knowledge, the first in-depth empirical study that measures how inputs individually interacts with software variability. To do so, we systematically explore the impact of inputs and configuration options on the performance properties of 8 software systems. This study reveals that inputs fed to software systems can indeed interact with their options in non monotonous ways, thus significantly impacting their performance properties. This observation questions the applicability of performance predictive models trained on only one input: are they still useful for other inputs? We then survey state-of-the-art papers on configurable systems to assess whether they address this kind of input sensitivity issue.

In summary, the contributions of this paper are as follows:

- To our best knowledge, the first in-depth empirical study that investigates the interactions between input data and configurations of 8 software systems;
- We show that **inputs fed to software systems interact with their configuration options in non monotonous ways**, thus changing performance of configurable systems and making their predictions difficult to automate;
- An analysis of how 64 state-of-the-art research papers on configurable systems address this problem in practice;
- Open science: a replication bundle that contains docker images, produced datasets of measurements and code.[1]

The remainder of this paper is organized as follows: Section 2 explains the problem of input sensitivity and the research questions addressed in this paper. Section 3 presents the experimental protocol. Section 4 details the results. Section 5 shows how researchers address input sensitivity. Section 6 discusses the implications of our work. Section 7 details threats to validity. Section 8 presents related work. Section 9 summarizes key insights of our paper.

Typographic Convention. For this paper, we adopt the following typographic convention: *emphasized* will be relative to a software system, slanted to its configuration options and underlined to its performance properties.

2 Problem Statement

2.1 Sensitivity to Inputs of Configurable Systems

Configuration options of software systems can have different effects on performance (*e.g.*, runtime), but so can the input data. For example, a configurable video encoder like *x264* can process many kinds of inputs (videos) in addition to offering options on how to encode. Our hypothesis is that there is an interplay between configuration options and input data: some (combinations of) options may have different effects on performance depending on input.

This sensitivity to inputs may have a strong impact on engineering and research work. Developers of configurable systems that process input data should be aware of this phenomenon and test their systems on a wide variety of inputs [74]. Similarly, researchers who develop learning algorithms or optimization techniques may want to benchmark them on a realistic set of inputs

[1] Available on Github:
https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/

to draw conclusions as general as possible on configuration spaces [13]. This notably concerns learning models that predict performance.

Researchers observed input sensitivity in multiple fields, such as SAT solvers [21, 98], compilation [16, 69], video encoding [57], data compression [46]. However, existing studies either consider a limited set of configurations (*e.g.*, only default configurations), a limited set of performance properties, or a limited set of inputs [1, 11, 22, 26, 51, 71, 82]. It limits some key insights about the input sensitivity of configurable systems.

This work details, to the best of our knowledge, the first systematic empirical study that analyzes the interactions between input data and configuration options for different configurable systems. Through three research questions introduced in the next section, we characterise the input sensitivity problem and explore how this can alter our understanding of software variability.

2.2 Research Questions

When a developer provides a default configuration for its software system, one should ensure it will perform at best for a large panel of inputs. That is, this configuration will be near-optimal whatever the input. Hence, an hidden assumption is that two performance distributions over two different inputs are somehow related and close. In its simplest form, there could be a linear relationship between these two distributions: they simply increase or decrease with each other.

> **RQ_1 - Do software performance stay consistent across inputs?**
> Are the performance distributions stable from one input to another? Are the rankings of performance the same for all inputs?

But software performance are influenced by the configuration options *e.g.*, the energy consumption [12]. An option is called influential for a performance when its values have a strong effect on this performance [17, 40]. For example, developers might wonder whether the option they add to a configurable software has an influence on its performance. However, is an option identified as influential for some inputs still influential for other inputs? If not, it would become both tedious and time-consuming to find influential options on a per-input basis. Besides, it is unclear whether activating an option is always worth it in terms of performance; an option could improve the overall performance while reducing it for few inputs. If so, users may wonder which options to enable to improve software performance based on their input data.

> **RQ_2 - Do configuration option's effects change with input data?**
> Do the configuration options have the same effects for all inputs? Is an influential option influential for all inputs? Do the effects of configuration options vary with input data?

RQ_1 and RQ_2 study how inputs affect (1) performance distributions and (2) the effects of different configuration options. However, the performance distributions could change in a negligible way, without affecting the software user's experience. Before concluding on the real impact of the input sensitivity, it is necessary to quantify how much this performance changes from one input to another.

> **RQ_3 - Can we ignore input sensitivity?** If we do, what is the loss in performance considering that all input data is the same and does not affect the software that processes it? Or, to put it more positively, what is the potential gain to tune a software system for its input data?

3 Experimental protocol

To answer these research questions, we have designed the following experimental protocol.

3.1 Data Collection

We first collect performance data of configurable systems that process inputs.

Protocol. Figure 2 depicts the step-by-step protocol we respect to measure performance of software systems. Each line of Table 1 should be read following Figure 2: System and Domain with Step 1; Commit with Step 2; Configs #C with Step 3; Inputs I and #I with Step 4; #M with Step 5; Performance(s) P with Step 6; Docker links a container for executing all the steps; Dataset links the results of the protocol *i.e.*, the datasets containing the performance measurements. Figure 2 shows in beige an example with the *x264* encoder. Hereafter, we provide details for each step of the protocol.

Steps 1 & 2 - Software Systems. We consider 8 software systems, open-source and well-known in various fields, that the literature already studied: *gcc* [69], *ImageMagick* [83], *lingeling* [34], *nodeJS* [36], *poppler* [55], *SQLite* [85], *x264* [39] and *xz* [91]. We choose these systems because they handle different types of input data, allowing us to draw as general conclusions as possible. For each software system, we use a unique private server with the same configuration running over the same operating system.[2] We download and compile a unique version of the system, related to the git *Commit* in Table 1. All performance are measured with this version of the software.

Step 3 - Configuration options C. To select the configuration options, we read the documentation of each system. We manually extracted the options affecting the performance of the system according to the documentation. We then sampled #C configurations by using random sampling [68]. We checked the uniformity of the different option values with a Kolmogorov-Smirnov test [56] applied to each configuration option.[3]

[2] The configurations of the running environments are available at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/replication/Environments.md

[3] Options and tests results are available at : https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/others/configs/sampling.md

Table 1: Subject ***Systems***. ***Domain*** the area of expertise using the system. ***Commit*** the git commit (*i.e.*, the version) of the system. ***Configs #C*** the number of configurations tested per system. ***Inputs I*** the type of input fed to the system. ***#I*** the number of inputs per system. ***#M*** the total number of measurements, #M = #I*#C. ***Performance(s) P*** the performance properties measured per system. ***Docker*** the links to the containers to replicate the measurements. ***Dataset*** the links to the measurements.

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

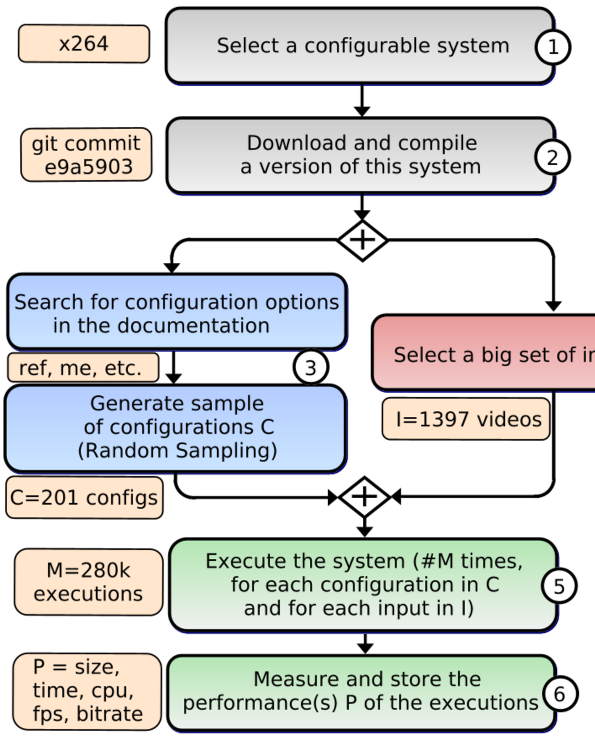


Fig. 2: Measuring performance - Protocol

Step 4 - Inputs I. For each system, we selected a different set of input data: for *gcc*, PolyBench v3.1 [73]; for *ImageMagick*, a sample of ImageNet [14] images (from 1.1 kB to 7.3 MB); for *lingeling*, the 2018 SAT competition's benchmark [34]; for *nodeJS*, its test suite; for poppler, the Trent Nelson's PDF Collection [64]; for *SQLite*, a set of generated TPC-H [70] databases (from 10 MB to 6 GB); for *x264*, the YouTube **U**ser **G**eneral **C**ontent dataset [94] of videos (from 2.7 MB to 39.7 GB); for *xz*, the Silesia corpus [15]. These are large, well-known and freely available datasets of inputs.

Steps 5 & 6 - Performance properties P. For each system, we systematically executed all the configurations of C on all the inputs of I. For the #M resulting executions, we measured as many performance properties as possible: for *gcc*, ctime and exec the times needed to compile and execute a program and the size of the binary; for *ImageMagick*, the time to apply a Gaussian blur [35] to an image and the size of the resulting image; for *lingeling*, the number of reductions and conflicts found in 10 seconds of execution; for *nodeJS*, the number of operations per second (ops) executed by the script; for *poppler*, the time needed to extract the images of the pdf, and the size of the images; for *SQLite*, the time needed to answer 15 different queries q1-q15; for *x264*, the size of the compressed video, the elapsed time, the cpu usage (percentage), the bitrate (the average amount of data encoded per second) and the average number of frames encoded per second (fps); for *xz*, the size of the compressed file, and the time needed to compress it.

Replication. To allow researchers to easily replicate the measurement process, we provide a docker container for each system (see the links in the *Docker* column of Table 1). We also publish the resulting datasets online (see the links in the *Dataset* column) and in the companion repository with additional replication details.[4]

For the next research questions, our results are computed with Python v3.7.6 and specific versions of data science libraries.[5]

3.2 Performance Correlations (RQ_1)

Based on the analysis of the data collected in Section 3.1, we can now answer the first research question: **RQ_1 - Do software performance stay consistent across inputs?** To check this hypothesis, we compute, analyze and compare the Spearman's rank-order correlation [45] of each couple of inputs for each system. It is appropriate in our case since all performance properties are quantitative variables measured on the same set of configurations.

Spearman correlations. The correlations are considered as a measure of similarity between the configurations' performance over two inputs. We compute the related *p*-values: a correlation whose *p*-value is higher than the chosen threshold 0.05 is considered as null. We use the Evans rule [20] to

[4] Guidelines for replication are available at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/replication/README.md

[5] The description of the python environment is available at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/replication/requirements.txt

interpret these correlations. In absolute value, we refer to correlations by the following labels; very low: 0-0.19, low: 0.2-0.39, moderate: 0.4-0.59, strong: 0.6-0.79, very strong: 0.8-1.00. A negative score tends to reverse the ranking of configurations. Very low or negative scores have practical implications: a good configuration for an input can very well exhibit bad performance for another input.

3.3 Effects of Options (RQ_2)

To understand how a performance model can change based on a given input, we next study how input data interact with configuration options. **$\mathbf{RQ_2}$ - Do configuration option's effects change with input data?** To assess the relative significance and effect of options, we use two well-known statistical methods [8, 77].

Random Forest Importances. The tree structure provides insights about the most essential options for prediction, because such a tree first splits w.r.t. options that provide the highest information gain. We use random forests [8], a vote between multiple decision trees: we can derive, from the forests trained on the inputs, estimates of the options importance. The computation of option importance is realized through the observation of the effect on random forest accuracy when randomly shuffling each predictor variable [58]. For a random forest, we consider that an option is influential if the median (on all inputs) of its option importance is greater than $\frac{1}{n_{opt}}$, where n_{opt} is the number of options considered in the dataset. This threshold represents the theoretic importance of options for a software having equally important options -inspired by the Kaiser rule [102].

Linear Regression Coefficients. The coefficients of an ordinary least square regression [77] weight the effect of configuration options. These coefficients can be positive (resp. negative) if a bigger (resp. lower) option value results in a bigger performance. Ideally, the sign of the coefficients of a given option should remain the same for all inputs: it would suggest that the effect of an option onto performance is stable. We also provide details about coefficients related to the interactions of options (*i.e.*, feature interactions [76, 89]) in RQ_2 results.

3.4 Impact of Input Sensitivity (RQ_3)

To complete this experimental protocol, we ask whether adapting the software to its input data is worth the cost of finding the right set of parameters *i.e.*, the concrete impact of input sensitivity. **$\mathbf{RQ_3}$ - Can we ignore input sensitivity?** To estimate how much we can lose, we first define two scenarios S_1 and S_2:

S_1: *Baseline.* In this scenario, we value input sensitivity and just train a simple performance model on an input - *i.e.*, the target input. We choose the best configuration according to the model, configure the related software with it and execute it on the target input.

S_2: *Ignoring input sensitivity.* In this scenario, let us pretend that we ignore the input sensitivity issue. We train a model related to a given input *i.e.*, the source input, and then predict the best configuration for this source input. If we ignore the issue of input sensitivity, we can easily reuse this model for any other input, including the target input defined in S_1. Finally, we execute the software with the configuration predicted by our model on the target input.

In this part, we systematically compare S_1 and S_2 in terms of performance for all inputs, all performance properties and all software systems. For S_1, we repeat the scenario ten times with different sources, uniformly chosen among other inputs and consider the average performance. For both scenarios, due to the imprecision of the learning procedure, the models can recommend sub-optimal configurations. Since this imprecision can alter the results, we consider an ideal case for both scenarios and assume that the performance models always recommend the best possible configuration.

Performance ratio. To compare S_1 and S_2, we use a performance ratio *i.e.*, the performance obtained in S_1 over the performance obtained in S_2. If the ratio is equal to 1, there is no difference between S_1 and S_2 and the input sensitivity does not exist. A ratio of 1.4 would suggest that the performance of S_1 is worth 1.4 times the performance of S_2; therefore, it is possible to gain up to $(1.4 - 1) * 100 = 40\%$ performance by choosing S_1 instead of S_2. We also report on the standard deviation of the performance ratio distribution. A standard deviation of 0 implies that for each input, we gain or lose the same proportion of performance when picking S_1 over S_2.

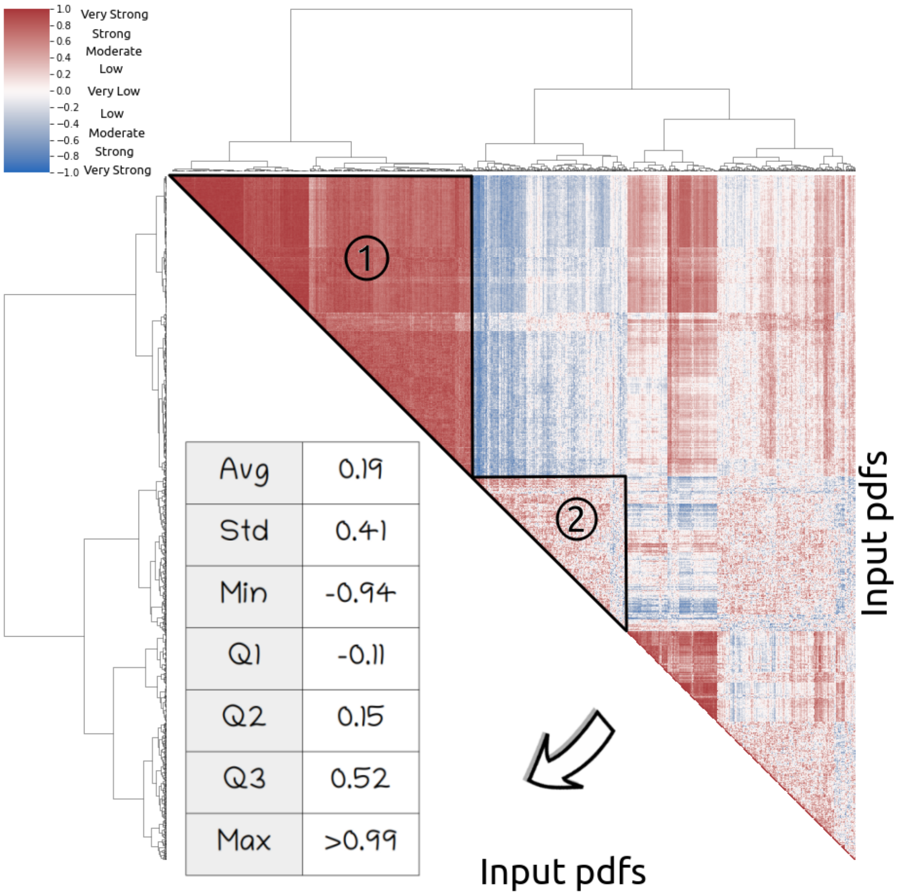
4 Results

We now present the results obtained by following the methodology defined in Section 3.

4.1 Performance Correlations (RQ_1)

We first explain the results of RQ_1 and their consequences on the *poppler* use case *i.e.*, an extreme case of input sensitivity, and then generalize to our other software systems.

Extract images of input pdfs with *poppler*. The content of pdf files fed to *poppler* may vary; the input pdf can be a 2-page report with a simple figure, a 10-page article or a 300-page picture book. Depending on this content, extracting the images embedded in those files can be quick or slow. Moreover, a user can adapt different configurations for the report and not for the book (or conversely), leading to different rankings in terms of extraction time.



Each square$_{(i,j)}$ represents the Spearman correlation between the time needed to extract the images of pdfs i and j. The color of this square respects the top-left scale: high positive correlations are red; low in white; negative in blue. Because we cannot describe each correlation individually, we added a table describing their distribution.

Fig. 3: Spearman correlogram - *poppler*, time.

In Figure 3, we depict the Spearman rank-order correlations, in terms of extraction time, between pairs of input pdfs fed to *poppler*. We also perform hierarchical clustering [42] on *poppler* data to gather inputs having similar time distributions and visually group correlated pdfs together.[6] Results suggest a positive correlation (see dark red cells), though there are pairs of inputs with lower (see white cells) and even negative (see dark blue cells) correlations. More than a quarter of the correlations between input pdfs are positive and at least moderate - third quartile Q3 greater than 0.52.

On the top-left part of the correlogram (see triangle ①), we even observe a first group of input pdfs that are highly correlated with each other - posi-

[6] Detailed RQ_1 results for other systems available at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/RQS/RQ1/RQ1.md

tively, strong or very strong. In this first group, the input pdfs have similar time rankings; their performance react the same way to the same configurations. However, this group of pdfs is uncorrelated (very low, low) or negatively correlated (moderate, strong and very strong) with the second group of pdfs - see the triangle ②. In this case, a single configuration working for the group ① should not be reused directly on a pdf of the group ②.

Meta-analysis. Over the 8 systems, we observe different cases:

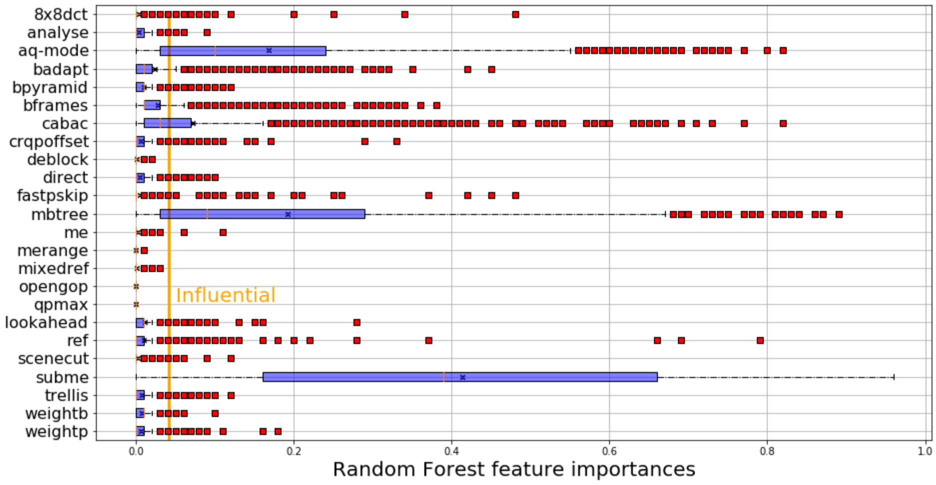
- There exist software systems not sensitive at all to inputs. In our experiment, *gcc*, *imagemagick* and *xz* present almost exclusively high and positive correlations between inputs *e.g.*, Q1 = 0.82 for the compressed size and *xz*. For these, un- or negatively-correlated inputs are an exception more than a rule.
- In contrast, there are software systems, namely *lingeling*, *nodeJS*, *SQLite* and *poppler*, for which performance distributions completely change and depend on input data *e.g.*, Q2 = 0.09 for *nodeJS* and ops, Q3 = 0.12 for *lingeling* and conflicts. For these, we draw similar conclusions as in the *poppler* case.
- In between, *x264* is only input-sensitive w.r.t. a performance property; it is for bitrate and size but not for cpu, fps and time *e.g.*, 0.29 as standard deviation for size and bitrate but 0.08 for the time.

> **RQ_1 - Do software performance stay consistent across inputs?** Performance distributions can change depending on inputs. Our systematic empirical study shows evidences about the existence of input sensitivity: (1) input sensitivity does not affect all systems; (2) input sensitivity may affect not the whole systems but some specific performance properties. So, without having scrutinized the input sensitivity of a system, one cannot develop techniques sensitive to this phenomenon.

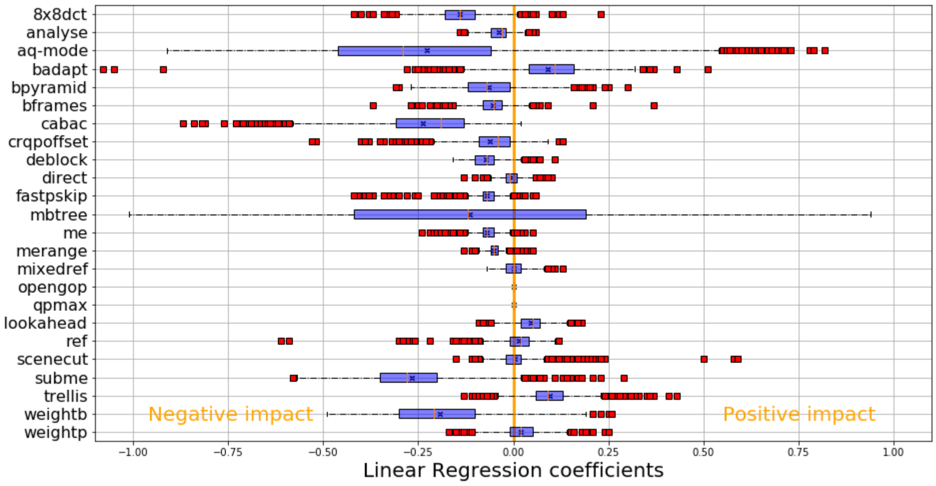
4.2 Effects of Options (RQ_2)

We first explain the results of RQ_2 and their concrete consequences on the bitrate of *x264* - an input-sensitive case, to then generalize to other software systems.

Encode input videos with *x264*. *x264* can encode different kinds of videos, such as an animation movie with many details, or a soccer game with large and monochromatic areas of grass. When encoding the soccer game, *x264* can use those fixed green areas to reduce the amount of data encoded per second (*i.e.*, the bitrate). In other words, configuration options aggregating pixels (*e.g.*, macro-block tree estimation mbtree) could both reduce the bitrate for the soccer game and increase the bitrate for the animation movie where nothing can be aggregated.



(a) Importance



(b) Effect

Fig. 4: Importance and effect of configuration options - *x264*, bitrate

Figures 4a and 4b report on respectively the boxplots of configuration options' feature importances and effects when predicting *x264*'s bitrate for all input videos.[7] Three options are strongly influential for a majority of videos on Figure 4a: subme, mbtree and aq-mode, but their importance can differ depending on input videos: for instance, the importance of subme is 0.83 for video #1365 and only 0.01 for video #40. Because influential options vary with input videos for *x264*, performance models and approaches based on *fea-*

[7] Detailed RQ_2 results for other systems available at: `https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/RQS/RQ2/RQ2.md`

ture selection [58] may not generalize well to all input videos. Most of the options have positive and negative coefficients on Figure 4b; thus, the specific effects of options heavily depend on input videos. It is also true for influential options: mbtree can have positive and negative (influential) effects on the bitrate *i.e.*, activating mbtree may be worth only for few input videos. The consequence is that one cannot reliably provide end-users with a unique *x264* default configuration whatever the input is.

Another interesting point is the link between RQ_1 and RQ_2 for *x264* and the bitrate; the more stable the effect of options in RQ_2, the more stable the distribution of performance in RQ_1. In fact, in a group of highly-correlated input videos (*e.g.*, like the group ① of pdfs in Figure 3, but for *x264*), the effect and importance of options are stable *i.e.*, the inputs all react the same way to the same options.[8] These different effects of influential options of *x264* may alter its encoding performance, thus explaining the different distributions pointed out in RQ_1. Under these circumstances, configuring the software system once per group of inputs is probably a reasonable solution for tackling input sensitivity.

Meta-analysis. For *gcc*, *imagemagick* and *xz*, the importances are quite stable. As an extreme case of stability, the importances of the compressed size for *xz* are exactly the same, except for two inputs. For these systems, the coefficients of linear regression mostly keep the same sign across inputs *i.e.*, the effects of options do not change with inputs. For input-sensitive software systems, we always observe high variations of options' effects (*lingeling*, *poppler* or *SQLite*), sometimes coupled to high variations of options' importances (*nodeJS*). For instance, the option format for *poppler* can have an importance of 0 or 1 depending on the input. For all software systems, there exists at least one performance property whose effects are not stable for all inputs *e.g.*, one input with negative coefficient and another with a positive coefficient. For *x264*, it depends on the performance property; for cpu, fps and time, the effect of influential options are stable for all inputs, while for the bitrate and the size, we can draw the conclusions previously presented in this section.

> **RQ_2 - Do configuration option's effects change with input data?** Different inputs lead to different configuration options' significance and effects. A set of influential options with changing effects can alter the distribution of performance, thus explaining RQ_1 results. Therefore, a configuration should not be fixed across inputs but evolve according to the input data fed to the system.

[8] See the detailed case of *x264* at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/others/x264_groups/x264_bitrate.md

4.3 Impact of Input Sensitivity (RQ_3)

This section presents the evaluation of RQ_3 w.r.t. the protocol of Section 3.4. In Table 2, we computed the performance ratios for the different software systems and their performance properties.[9]

Table 2: Performance ratio distributions across inputs, for different software systems and different performance properties. In lines, *Avg* the average performance ratio. *Std* the standard deviation. 5^{th} the 5^{th} percentile. *Q1* the first quartile. *Q2* the median. *Q3* the third quartile. 95^{th} the 95^{th} percentile. Due to space constraints, we arbitrarily select few performance properties.

System	***gcc***			***lingeling***		***nodeJS***	***poppler***	
Perf. P	ctime	exec	size	confl	reduc	ops	size	time
Avg	1.08	1.13	1.27	2.11	1.38	1.73	1.56	2.69
Std	0.07	0.07	0.36	2.6	0.79	1.88	1.27	3.72
5^{th}	1.0	1.05	1.01	1.02	1.0	1.01	1.0	1.03
Q1	1.01	1.11	1.04	1.05	1.04	1.08	1.0	1.14
Q2	1.08	1.12	1.16	1.14	1.11	1.16	1.07	1.38
Q3	1.11	1.14	1.32	1.47	1.25	1.54	1.51	2.22
95^{th}	1.2	1.2	1.97	8.05	2.79	4.22	3.85	10.11

System	***SQLite***			***x264***					***xz***	
Perf. P	q1	q12	q14	cpu	etime	fps	bitrate	size	size	time
Avg	1.03	1.08	1.07	1.42	1.43	1.1	1.11	1.11	1.0	1.08
Std	0.02	0.05	0.05	1.27	1.45	0.14	0.13	0.13	0.0	0.06
5^{th}	1.01	1.01	1.01	1.05	1.05	1.02	1.01	1.02	1.0	1.0
Q1	1.02	1.03	1.03	1.12	1.12	1.04	1.03	1.05	1.0	1.02
Q2	1.03	1.07	1.07	1.21	1.21	1.06	1.07	1.08	1.0	1.07
Q3	1.04	1.11	1.09	1.38	1.37	1.1	1.15	1.12	1.0	1.11
95^{th}	1.08	1.17	1.16	2.11	2.11	1.25	1.32	1.28	1.0	1.2

For software systems whose performance are stable across inputs (*gcc*, *imagemagick* and *xz*), there are few differences between inputs. For instance, for the output size of *xz*, there is no variation between scenarios S_1 (*i.e.*, using the best configuration) and S_2 (*i.e.*, reusing a the best configuration of a given input for another input): all performance ratios (*i.e.*, performance S_1 over performance S_2) are equals to 1 whatever the input.

For input-sensitive software systems (*lingeling*, *nodeJS*, *SQLite* and *poppler*), changing the configuration can lead to a negligible change in a few cases. For instance, for the time to answer the first query q1 with *SQLite*, the third quartile is 1.04; in this case, *SQLite* is sensitive to inputs, but its variations of performance -less than 4 %- do not justify the complexity of tuning the software. But it can also be a huge change; for *lingeling* and solved conflicts, the 95^{th} percentile ratio is equal to 8.05 *i.e.*, a factor of 8 between S_1 and S_2. It goes up to a ratio of 10.11 for *poppler*'s extraction time: there exists an input pdf for which extracting its images is ten times slower when reusing a configuration, compared to the best one (*i.e.*, the fastest).

[9] Detailed RQ_3 results for other performance properties available at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/RQS/RQ3/RQ3.md

In between, *x264* is a complex case. For its low input-sensitive performance (*e.g.*, cpu and etime), it moderately impacts the performance when reusing a configuration from one input to another - average ratios at resp. 1.42 and 1.43. In this case, the rankings of performance do not change a lot with inputs, but a small ranking change does make the difference in terms of performance.

On the contrary, for the input-sensitive performance (*e.g.*, the bitrate), there are few variations of performance: we can lose $1 - \frac{1}{1.11} \simeq 9\,\%$ of bitrate in average. In this case, it is up to the compression experts to decide; if losing up to $1 - \frac{1}{1.32} \simeq 24\,\%$ of bitrate is acceptable, then we can ignore input sensitivity. Otherwise, we should consider tuning *x264* for its input video.

> **RQ_3 - Can we ignore input sensitivity?** There exist input-sensitive cases for which the difference of performance does not justify to consider the input sensitivity *e.g.*, 5 % change is probably negligible. However, performance can be multiplied up to a ratio of 10 if we tune other systems for their input data: we cannot ignore it.

5 Sensitivity to Inputs in Research

In this section, we explore the significance of the input sensitivity problem in research. Do researchers know the issue of input sensitivity? How do they deal with inputs in their papers? Is the interaction between software configurations and input sensitivity a well-known issue?

5.1 Experimental Protocol

First, we aim at gathering research papers that actually predict performance of configurable systems *i.e.*, with a performance model [28].

Gather research papers. We focused on the publications of the last ten years. To do so, we analyzed the papers published (strictly) after 2011 from the survey of Pereira *et al.* [67] - published in 2019. We completed those papers with more recent papers (2019-2021), following the same procedure as in [67]. We have only kept research work that trained performance models on software systems.

Search for input sensitivity. We read each selected paper and answered four different questions: Q-A. Is there a software system processing input data in the study? If not, the impact of input sensitivity in the existing research work would be relatively low. The idea of this research question is to estimate the proportion of the performance models that could be affected by input sensitivity. Q-B. Does the experimental protocol include several inputs? If not, it would suggest that the performance model only captures a partial truth, and might not generalize for other inputs fed to the software system. Q-C. Is the problem of input sensitivity mentioned *e.g.*, in threat? This question aims to state whether researchers are aware of the input sensitivity issue, and estimate the proportion of the papers that mention it as a potential threat to validity. Q-D. Does the paper propose a solution to generalize the performance model across inputs? Finally, we check whether the paper proposes a solution managing input sensitivity *i.e.*, if the proposed approach could be adapted to our problem and predict a near-optimal configuration for any input. The results were obtained by one author and validated by all other co-authors.

5.2 How do Research Papers address Input Sensitivity?

Table 3 lists the 64 research papers we identified following this protocol, as well as their individual answers to Q-A→Q-D. A checked cell indicates that the answer to the corresponding question (column) for the corresponding paper (line) is *yes*. Since answering Q-B, Q-C or Q-D only makes sense if Q-A is checked, we grayed and did not consider Q-B, Q-C and Q-D if the answer of Q-A is *no*. We also provide full references and detailed justifications in the companion repository.[10] We now comment the average results for each question:

Table 3: Input sensitivity in research. Paper identifier ***ID*** in the list. ***Authors*** of the paper. ***Conference*** in which the paper was accepted. ***Year*** of publication of the paper. ***Title*** of the paper. ***Q-A.*** Is there a software system processing input data in the study? ***Q-B.*** Does the experimental protocol include several inputs? ***Q-C.*** Is the problem of input sensitivity mentioned *e.g.*, in threat? ***Q-D.*** Does the paper propose a solution to generalize the performance model across inputs? Due to space limitation, we do not justify the answers directly in the paper, see the companion repository (file *results/RQS/RQ4/RQ4.md*) for justifications.

ID	*Authors*	*Conference*	*Year*	*Title*	*Q-A*	*Q-B*	*Q-C*	*Q-D*
1	Guo *et al.* [29]	ESE	2017	Data-efficient performance learning for configurable systems	X			
2	Jamshidi *et al.* [41]	SEAMS	2017	Transfer learning for improving model predictions [...]	X	X	X	
3	Jamshidi *et al.* [39]	ASE	2017	Transfer learning for performance modeling of configurable [...]	X	X	X	X
4	Oh *et al.* [65]	ESEC/FSE	2017	Finding near-optimal configurations in product lines by [...]	X			
5	Kolesnikov *et al.* [47]	SoSyM	2018	Tradeoffs in modeling performance of highly configurable [...]	X			
6	Nair *et al.* [61]	ESEC/FSE	2017	Using bad learners to find good configurations	X	X		
7	Nair *et al.* [63]	TSE	2018	Finding Faster Configurations using FLASH	X	X	X	
8	Murwantara *et al.* [59]	iiWAS	2014	Measuring Energy Consumption for Web Service Product [...]	X	X	X	X

[10] The list of papers can be consulted at https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/RQS/RQ4/RQ4.md

ID	*Authors*	*Conference*	*Year*	*Title*	*Q-A*	*Q-B*	*Q-C*	*Q-D*
9	Temple *et al.* [87]	SPLC	2016	Using Machine Learning to Infer Constraints for Product Lines				
10	Temple *et al.* [85]	IEEE Soft.	2017	Learning Contextual-Variability Models	X		X	
11	Valov *et al.* [91]	ICPE	2017	Transferring performance prediction models across different [...]	X		X	X
12	Weckesser *et al.* [96]	SPLC	2018	Optimal reconfiguration of dynamic software product [...]				
13	Acher *et al.* [2]	VaMoS	2018	VaryLATEX: Learning Paper Variants That Meet Constraints	X	X		X
14	Sarkar *et al.* [76]	ASE	2015	Cost-Efficient Sampling for Performance Prediction of [...]	X			
15	Temple *et al.* [84]	Report	2018	Towards Adversarial Configurations for Software Product Lines				
16	Nair *et al.* [62]	ASE	2018	Faster Discovery of Faster System Configurations with [...]	X			
17	Siegmund *et al.* [79]	ESEC/FSE	2015	Performance-Influence Models for Highly Configurable Systems	X			
18	Valov *et al.* [89]	SPLC	2015	Empirical comparison of regression methods for [...]	X			
19	Zhang *et al.* [99]	ASE	2015	Performance Prediction of Configurable Software Systems [...]	X		X	
20	Kolesnikov *et al.* [48]	ESE	2019	On the relation of control-flow and performance feature [...]	X			
21	Couto *et al.* [12]	SPLC	2017	Products go Green: Worst-Case Energy Consumption [...]	X		X	
22	Van Aken *et al.* [92]	SIGMOD	2017	Automatic Database Management System Tuning Through [...]	X	X	X	X
23	Kaltenecker *et al.* [44]	ICSE	2019	Distance-based sampling of software configuration spaces	X			
24	Jamshidi *et al.* [40]	ESEC/FSE	2018	Learning to sample: exploiting similarities across [...]	X	X	X	X
25	Jamshidi *et al.* [38]	MASCOTS	2016	An Uncertainty-Aware Approach to Optimal Configuration of [...]	X	X	X	
26	Lillacka *et al.* [53]	Soft. Eng.	2013	Improved prediction of non-functional properties in Software [...]	X	X	X	X
27	Zuluaga *et al.* [101]	JMLR	2016	ε-pal: an active learning approach [...]	X	X		
28	Amand *et al.* [6]	VaMoS	2019	Towards Learning-Aided Configuration in 3D Printing [...]	X	X	X	
29	Alipourfard *et al.* [4]	NSDI	2017	Cherrypick: Adaptively unearthing the best cloud [...]	X	X	X	
30	Saleem *et al.* [75]	TSC	2015	Personalized Decision-Strategy based Web Service Selection [...]	X	X		
31	Zhang *et al.* [100]	SPLC	2016	A mathematical model of performance-relevant [...]	X			
32	Ghamizi *et al.* [25]	SPLC	2019	Automated Search for Configurations of Deep Neural [...]	X	X	X	
33	Grebhahn *et al.* [27]	CPE	2017	Performance-influence models of multigrid methods [...]				
34	Bao *et al.* [7]	ASE	2018	AutoConfig: Automatic Configuration Tuning for Distributed [...]	X	X		
35	Guo *et al.* [28]	ASE	2013	Variability-aware performance prediction: A statistical [...]	X			
36	Švogor *et al.* [103]	IST	2019	An extensible framework for software configuration optim[...]	X	X		

ID	Authors	Conference	Year	Title	Q-A	Q-B	Q-C	Q-D
37	El Afia *et al.* [3]	CloudTech	2018	Performance prediction using support vector machine for the [...]	X	X		
38	Ding *et al.* [16]	PLDI	2015	Autotuning algorithmic choice for input sensitivity	X	X	X	X
39	Duarte *et al.* [19]	SEAMS	2018	Learning Non-Deterministic Impact Models for Adaptation	X	X	X	X
40	Thornton *et al.* [88]	KDD	2013	Auto-WEKA: Combined selection and hyperparameter [...]	X	X	X	
41	Siegmund *et al.* [80]	ICSE	2012	Predicting performance via automated feature-inter[...]	X	X	X	
42	Siegmund *et al.* [81]	SQJ	2012	SPL Conqueror: Toward optimization of non-functional [...]	X	X		
43	Westermann *et al.* [97]	ASE	2012	Automated inference of goal-oriented performance prediction [...]	X	X		
44	Velez *et al.* [93]	ICSE	2021	White-Box Analysis over Machine Learning: Modeling [...]	X	X		
45	Pereira *et al.* [5]	ICPE	2020	Sampling Effect on Performance Prediction of Configurable [...]	X	X	X	
46	Shu *et al.* [78]	ESEM	2020	Perf-AL: Performance prediction for configurable software [...]	X			
47	Dorn *et al.* [18]	ASE	2020	Mastering Uncertainty in Performance Estimations of [...]	X			
48	Kaltenecker *et al.* [43]	IEEE Soft.	2020	The Interplay of Sampling and Machine Learning for Software [...]	X			
49	Krishna *et al.* [49]	TSE	2020	Whence to Learn? Transferring Knowledge in Configurable [...]	X	X	X	X
50	Weber *et al.* [95]	ICSE	2021	White-Box Performance-Influence Models: A Profiling [...]	X	X		
51	Mühlbauer *et al.* [60]	ASE	2020	Identifying Software Performance Changes Across Variants [...]	X	X		
52	Han *et al.* [32]	Report	2020	Automated Performance Tuning for Highly-Configurable [...]	X	X		
53	Han *et al.* [33]	ICPE	2021	ConfProf: White-Box Performance Profiling of Configuration [...]	X		X	
54	Valov *et al.* [90]	ICPE	2020	Transferring Pareto Frontiers across Heterogeneous Hardware [...]	X			X
55	Liu *et al.* [54]	CF	2020	Deffe: a data-efficient framework for performance [...]	X	X	X	X
56	Fu *et al.* [23]	NSDI	2021	On the Use of ML for Blackbox System Performance Prediction	X	X	X	X
57	Larsson *et al.* [50]	IFIP	2021	Source Selection in Transfer Learning for Improved Service [...]	X	X	X	X
58	Chen *et al.* [9]	ICSE	2021	Efficient Compiler Autotuning via Bayesian Optimization	X	X	X	
59	Chen *et al.* [10]	SEAMS	2019	All Versus One: An Empirical Comparison on Retrained [...]	X	X		
60	Ha *et al.* [30]	ICSE	2019	DeepPerf: Performance Prediction for Configurable Software [...]	X			
61	Pei *et al.* [66]	Report	2019	DeepXplore: automated whitebox testing of deep learning systems	X	X		
62	Ha *et al.* [31]	ICSME	2019	Performance-Influence Model for Highly Configurable [...]	X			
63	Iorio *et al.* [37]	CloudCom	2019	Transfer Learning for Cross-Model Regression in Performance [...]	X	X	X	X
64	Koc *et al.* [72]	ASE	2021	SATune: A Study-Driven Auto-Tuning Approach for [...]	X	X	X	X
Total					60	38	28	16

Q-A. Is there a software system processing input data in the study? Of the 64 papers, 60 (94 %) consider at least one configurable system processing inputs. This large proportion gives credits to input sensitivity and its potential impact on research work.

Q-B. Does the experimental protocol include several inputs? 63 % of the research work answering *yes* to Q-A include different inputs in their protocol. But what about the other 37 %? It is understandable not to consider several inputs because of the cost of measurements. However, if we reproduce all experiments of Table 3 using other input data, will we draw the same conclusions for each paper? Based on the results of $RQ_1 \rightarrow RQ_3$, we encourage all researchers to consider at least a set of well-chosen inputs in their protocol *e.g.*, an input per group, as shown in RQ_1. We give an example of such a set for *x264* in Section 6.

Q-C. Is the problem of input sensitivity mentioned *e.g.*, in threat? Only half (47 %) of the papers mention the issue of input sensitivity, mostly without naming it or using a domain-specific keyword *e.g.*, workload variation [91]. For the other half, we cannot guarantee with certainty that input sensitivity concerns all papers. But we shed light on this issue: ignoring input sensitivity can prevent the generalization of performance models across inputs. This is especially true for the 37 % of papers answering *no* to Q-B *i.e.*, considering one input per system: only 14 % of these research works mention it in their publication.

Q-D. Does the paper propose a solution to generalize the performance model across inputs? We identified 16 papers [2, 16, 19, 23, 37, 39, 40, 49, 50, 53, 54, 59, 72, 90–92] proposing contributions that may help in better managing the input sensitivity problem, and that should be adapted and tested (*e.g.*, with our data) to evaluate their ability to support this problem.

> **Conclusion.** While half of the research articles mention input sensitivity, few actually address it, and most often on a single system and domain. Input sensitivity can affect multiple research works and questions their practical relevance for a field deployment.

6 Impact for researchers and research opportunities

This section discusses the implications of our work.

Impacts for researchers. We warn researchers that the effectiveness of learning strategies for a given configurable system can be biased by the inputs and the performance property used. That is, a sampling strategy, a prediction or optimisation algorithm, or a transfer technique may well be highly accurate for an input and still inaccurate for others. Most of the studies neglect either inputs or configurations, which is perfectly understandable owing to the investments required. However, the scientific community should be extremely careful with this input sensitivity issue. In view of the results of our study, new problems deserve to be tackled with associated challenges. We detail some of them hereafter.

Sampling configurations. With the promise to select a small and representative sample set of valid configurations, several sampling strategies have been devised in the last years [5, 43, 67] (*e.g.*, random sampling, *t*-wise sampling, distance-based sampling). As recently reported in other experimental settings [5, 43], finding the most effective combinations of sampling and learning strategies is an open problem. Input sensitivity further exacerbates the problem. We conjecture that some strategies for sampling configurations might be effective for specific inputs and performance properties. Pereira *et al.* [5] actually provided preliminary evidence on *x264* for 19 input videos and two performance properties. Our results show and confirm that the importance of options and their interactions is indeed sensitive to the input (see RQ_2), thus suggesting that some sampling strategies may not always capture them. An open issue is thus to find sampling strategies that are effective for any input.

Tuning and performance prediction. Numerous works aim to find optimal configurations or predict the performance of an arbitrary configuration. However, our empirical results show that the best configuration can be differently ranked (see RQ_1) depending on an input. The tuning or the prediction cannot be reused as such (see RQ_3) but should be redone or adapted whenever a system processes a new input. To illustrate this, we present a minimal example using SPLConqueror [81]: we train two performance models predicting the encoding sizes of two different input videos fed to *x264* and show that the two related models do not share any common (interaction of) option.[11] So, an open challenge is to deliver algorithms and practical tools capable of tuning a system whatever the input. Another issue is to reduce the cost of training models for each input (*e.g.*, through sampling or transfer learning).

Understanding of configurable systems. Understanding the effects of options and their interactions is hard for developers and users yet crucial for maintaining, debugging or configuring a software system. Some works (*e.g.*, [95]) have proposed to build performance models that are interpretable and capable of communicating the influence of individual options and interactions on performance (possibly back to the code). Our empirical results show that performance models, options and their interactions are sensitive to inputs (see RQ_2). A first open issue is to communicate when and how options together with input data interact and influence performance. Another challenge is to identify a minimal set of representative inputs in such a way a configurable system can be observed and performance models learnt.

Recommendations for researchers and practitioners. Given the state of the art and the open problems to be addressed, there is no complete solution that can be systematically employed. However, we can give two recommendations: *(1) Detecting input sensitivity.* As practitioners dealing with new inputs, we first have to to determine whether the software under study is input-

[11] See the performance models for the first and the second input videos.

sensitive w.r.t. the performance property of interest. If the input sensitivity is negligible (see RQ_3), we can use a single model to predict the performance of the software system. If not, measurements over multiple inputs are needed. *(2) Selecting representative inputs.* To reduce the cost of measurements, the ideal would be to select a set of input data, both representative of the usage of the system and cheap to measure. We believe our work can be helpful here. On the *x264* case study, for the bitrate, we isolate four encoding groups of input videos (action movie - big resolution - still image - standard). Within a group, the videos share common properties, and *x264* processes them in the same way *i.e.*, same performance distributions (RQ_1), same options' effects (RQ_2) and a negligible effect of input sensitivity (RQ_3). In the companion repository, we propose to reduce the dataset of 1397 input videos [94] to a subset of 8 videos, selecting 2 cheap videos in each group of performance.[12] Automating this grouping could drastically reduce the cost of measuring configurations' performance over inputs.

7 Threats to Validity

This section discusses the threats to validity related to our protocol.

Construct validity. Due to resource constraints, we did not include all the options of the configurable systems in the experimental protocol. We may have forgotten configuration options that matter when predicting the performance of our configurable systems. However, we consider features that impact the performance properties according to the documentation, which is sufficient to show the existence of the input sensitivity issue.

Internal Validity. First, our results can be subject to measurement bias. We alleviated this threat by making sure only our experiment was running on the server we used to measure the performance of software systems. It has several benefits: we can guarantee we use similar hardware (both in terms of CPU and disk) for all measurements; we can control the workload of each machine (basically we force the machine to be used only by us); we can avoid networking and I/O issues by placing inputs on local folders. But it could also represent a threat: our experiments may depend on the hardware and operating system. The measurement process is launched via docker containers. If this aims at making this work reproducible, this can also alter the results of our experiment. Because of the amount of resources needed to compute all the measures, we did not repeat the process of Figure 2 several times per system. We consider that the large number of inputs under test overcomes this threat. Moreover, related work (*e.g.*, [5] for *x264*) has shown that inputs often maintain stable performance between different launches of the same configuration. Finally, the measurement process can also suffer from a lack of inputs. To limit this problem, we took relevant dataset of inputs produced and widely used in

[12] See the resulting benchmark and its construction at: https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/results/others/x264_groups/x264_bitrate.md

their field. For RQ_1-RQ_3, executing our code with another python environment may lead to slightly different conclusions. For RQ_3, we consider oracles when predicting the best configurations for both scenarios, thus neglecting the imprecision of performance models: these results might change on a real-world case. In Section 5, our results are subject to the selection of research papers: since we use and reproduce [67], we face the same threats to validity.

External Validity. A threat to external validity is related to the used case studies and the discussion of the results. Because we rely on specific systems and interesting performance properties, the results may be subject to these systems and properties. To reduce this bias, we selected multiple configurable systems, used for different purposes in different domains.

8 Related Work

In this section, we discuss other related work (see also Section 5).

Workload Performance Analysis. On the one hand some work have been addressing the performance analysis of software systems [11, 22, 26, 51, 71, 82] depending on different input data (also called workloads or benchmarks), but all of them only considered a rather limited set of configurations. On the other hand, as already discussed in Section 5, works and studies on configurable systems usually neglect input data (*e.g.*, using a unique video for measuring the configurations of a video encoder). In this paper, we combined both dimensions by performing an in-depth, controlled study of several configurable systems to make it vary in the large, both in terms of configurations and inputs. In contrast to research papers considering multiple factors of the executing environment in the wild [40, 91], we concentrated on inputs and software configurations only, which allowed us to draw reliable conclusions regarding the specific impact of inputs on software variability.

Performance Prediction. Research work have shown that machine learning could predict the performance of configurations [28,76,89,99]. These works measure the performance of a configuration sample under specific settings to then build a model capable of predicting the performance of any other configuration, *i.e.*, a performance model. Numerous works have proposed to model performance of software configurations, with several use-cases in mind for developers and users of software systems: the maintenance and understanding of configuration options and their interactions [79], the selection of an optimal configuration [24, 63, 65], the automated specialization of configurable systems [85, 86]. Input sensitivity complicates their task; since inputs affect software performance, it is yet a challenge to train reusable performance prediction models *i.e.*, that we could apply on multiple inputs.

Input-aware tuning. The input sensitivity issue has been partly considered in some specific domains (SAT solvers [21,98], compilation [16,69], video encoding [57], data compression [46], *etc.*). It is unclear whether these ad hoc solutions are cost-effective. As future work, we plan to systematically assess domain-specific techniques as well as generic, domain-agnostic approach (*e.g.*, transfer learning) using our dataset. Furthermore, the existence of a general solution applicable to all domains and software configurations is an open question. For example, is it always possible and effective to extract input properties for all kinds of inputs?

Input Data and other Variability Factors. Most of the studies support learning models restrictive to specific static settings (*e.g.*, inputs, hardware, and version) such that a new prediction model has to be learned from scratch once the environment change [67]. Jamshidi *et al.* [39] conducted an empirical study on four configurable systems (including *SQLite* and *x264*), varying software configurations and environmental conditions, such as hardware, input, and software versions. But without isolating the individual effect of input data on software configurations, it is challenging to understand the existing interplay between the inputs and any other variability factor *e.g.*, the hardware.

9 Conclusion

We conducted a large study over the inputs fed to 8 configurable systems that shows the significance of the input sensitivity problem on performance properties. We deliver one main message: **inputs interact with configuration options in non monotonous ways, thus making it difficult to (automatically) configure a system**. It appears that inputs can significantly change the performance of the configurable systems up to the point some options' values have an opposite effect depending on the input. Ignoring this lesson could lead to the learning of inaccurate performance prediction models and ineffective configuration recommendations for developers and end-users.

As future work, it is an open challenge to solve the issue of input sensitivity when predicting, optimising or understanding configurable systems. We encourage researchers to confront both existing methods of the literature and future ideas with our data.

References

1. Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne Aaron: A Large-Scale Comparison of x264, x265, and libvpx – a Sneak Peek. netflix-study (2016)
2. Acher, M., Temple, P., Jézéquel, J.M., Galindo, J.A., Martinez, J., Ziadi, T.: Varylatex: Learning paper variants that meet constraints. In: Proc. of VAMOS'18, p. 83–88 (2018). DOI https://doi.org/10.1145/3168365.3168372
3. Afia, A.E., Sarhani, M.: Performance prediction using support vector machine for the configuration of optimization algorithms. In: Proc. of CloudTech'17, pp. 1–7 (2017). DOI 10.1109/CloudTech.2017.8284699
4. Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: Proc. of NSDI'17, p. 469–482 (2017)
5. Alves Pereira, J., Acher, M., Martin, H., Jézéquel, J.M.: Sampling effect on performance prediction of configurable systems: A case study. In: Proc. of ICPE'20, p. 277–288 (2020)

6. Amand, B., Cordy, M., Heymans, P., Acher, M., Temple, P., Jézéquel, J.M.: Towards learning-aided configuration in 3d printing: Feasibility study and application to defect prediction. In: Proc. of VAMOS'19 (2019). DOI 10.1145/3302333.3302338. URL https://doi.org/10.1145/3302333.3302338
7. Bao, L., Liu, X., Xu, Z., Fang, B.: Autoconfig: Automatic configuration tuning for distributed message systems. In: Proc. of ASE'18, p. 29–40 (2018). DOI 10.1145/3238147.3238175. URL https://doi.org/10.1145/3238147.3238175
8. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
9. Chen, J., Xu, N., Chen, P., Zhang, H.: Efficient compiler autotuning via bayesian optimization. In: Proc. of ICSE'21, pp. 1198–1209 (2021). DOI 10.1109/ICSE43902.2021.00110
10. Chen, T.: All versus one: An empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software. In: Proc. of SEAMS'19, pp. 157–168 (2019). DOI 10.1109/SEAMS.2019.00029
11. Coppa, E., Demetrescu, C., Finocchi, I., Marotta, R.: Estimating the Empirical Cost Function of Routines with Dynamic Workloads. In: Proc. of CGO'14, p. 230:239 (2014). DOI 10.1145/2581122.2544143. URL http://doi.acm.org/10.1145/2581122.2544143
12. Couto, M., Borba, P., Cunha, J., Fernandes, J.a.P., Pereira, R., Saraiva, J.a.: Products go green: Worst-case energy consumption in software product lines. In: Proc. of SPLC'17, p. 84–93 (2017). DOI 10.1145/3106195.3106214. URL https://doi.org/10.1145/3106195.3106214
13. de Oliveira Neto, F.G., Torkar, R., Feldt, R., Gren, L., Furia, C.A., Huang, Z.: Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. Journal of Systems and Software **156**, 246–267 (2019). DOI https://doi.org/10.1016/j.jss.2019.07.002. URL https://www.sciencedirect.com/science/article/pii/S0164121219301451
14. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). DOI 10.1109/CVPR.2009.5206848
15. Deorowicz, S.: Silesia corpus. (2003). URL http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia
16. Ding, Y., Ansel, J., Veeramachaneni, K., Shen, X., O'Reilly, U.M., Amarasinghe, S.: Autotuning algorithmic choice for input sensitivity. In: ACM SIGPLAN Notices, vol. 50, pp. 379–390. ACM (2015)
17. Dorn, J., Apel, S., Siegmund, N.: Generating attributed variability models for transfer learning. In: Proc. of VAMOS'20 (2020)
18. Dorn, J., Apel, S., Siegmund, N.: Mastering uncertainty in performance estimations of configurable software systems. In: Proc. of ASE'20, p. 684–696 (2020). DOI 10.1145/3324884.3416620. URL https://doi.org/10.1145/3324884.3416620
19. Duarte, F., Gil, R., Romano, P., Lopes, A., Rodrigues, L.: Learning non-deterministic impact models for adaptation. In: Proc. of SEAMS'18, p. 196–205 (2018). DOI 10.1145/3194133.3194138. URL https://doi.org/10.1145/3194133.3194138
20. Evans, J.D.: Straightforward statistics for the behavioral sciences. Brooks/Cole Publishing Company (1996)
21. Falkner, S., Lindauer, M., Hutter, F.: Spysmac: Automated configuration and performance analysis of sat solvers. In: Proc. of SAT'15, pp. 215–222 (2015)
22. FathyAtlam, H., Attiya, G., El-Fishawy, N.: Comparative study on CBIR based on color feature. International Journal of Computer Applications **78**(16), 9–15 (2013). DOI 10.5120/13605-1387. URL https://doi.org/10.5120/13605-1387
23. Fu, S., Gupta, S., Mittal, R., Ratnasamy, S.: On the use of ML for blackbox system performance prediction. In: Proc. of NSDI'21, pp. 763–784 (2021). URL https://www.usenix.org/conference/nsdi21/presentation/fu
24. Fu, W., Menzies, T.: Easy over hard: A case study on deep learning. In: Proc. of ESEC-FSE'17, p. 49–60 (2017). DOI 10.1145/3106237.3106256. URL https://doi.org/10.1145/3106237.3106256
25. Ghamizi, S., Cordy, M., Papadakis, M., Traon, Y.L.: Automated search for configurations of convolutional neural network architectures. In: Proc. of SPLC'19, p. 119–130 (2019). DOI 10.1145/3336294.3336306. URL https://doi.org/10.1145/3336294.3336306

26. Goldsmith, S.F., Aiken, A.S., Wilkerson, D.S.: Measuring empirical computational complexity. In: Proc. of ESEC-FSE'07, p. 395–404 (2007)
27. Grebhahn, A., Rodrigo, C., Siegmund, N., Gaspar, F., Apel, S.: Performance-influence models of multigrid methods: A case study on triangular grids. Concurrency and Computation: Practice and Experience **29** (2017)
28. Guo, J., Czarnecki, K., Apely, S., Siegmundy, N., Wasowski, A.: Variability-aware performance prediction: A statistical learning approach. In: Proc. of ASE'13, p. 301–311 (2013). DOI 10.1109/ASE.2013.6693089. URL `https://doi.org/10.1109/ASE.2013.6693089`
29. Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., Yu, H.: Data-efficient performance learning for configurable systems. Empirical Software Engineering **23**(3), 1826–1867 (2017)
30. Ha, H., Zhang, H.: Deepperf: Performance prediction for configurable software with deep sparse neural network. In: Proc. of ICSE'19, pp. 1095–1106 (2019). DOI 10.1109/ICSE.2019.00113
31. Ha, H., Zhang, H.: Performance-influence model for highly configurable software with fourier learning and lasso regression. In: Proc. of ICSME'19, pp. 470–480 (2019)
32. Han, X., Yu, T.: Automated performance tuning for highly-configurable software systems. arXiv preprint arXiv:2010.01397 (2020)
33. Han, X., Yu, T., Pradel, M.: Confprof: White-box performance profiling of configuration options. In: Proc. of ICPE'12, p. 1–8 (2021)
34. Heule, M., Järvisalo, M., Suda, M. (eds.): Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions, *Department of Computer Science Series of Publications B*, vol. B-2018-1. University of Helsinki, Finland (2018)
35. Hummel, R.A., Kimia, B., Zucker, S.W.: Deblurring gaussian blur. Computer Vision, Graphics, and Image Processing **38**(1), 66–80 (1987)
36. Incerto, E., Tribastone, M., Trubiani, C.: Software performance self-adaptation through efficient model predictive control. In: Proc. of ASE'17, pp. 485–496 (2017). DOI 10.1109/ASE.2017.8115660
37. Iorio, F., Hashemi, A.B., Tao, M., Amza, C.: Transfer learning for cross-model regression in performance modeling for the cloud. In: Proc. of CloudCom'19, pp. 9–18 (2019). DOI 10.1109/CloudCom.2019.00015
38. Jamshidi, P., Casale, G.: An uncertainty-aware approach to optimal configuration of stream processing systems. CoRR (2016). URL `http://arxiv.org/abs/1606.06543`
39. Jamshidi, P., Siegmund, N., Velez, M., Kästner, C., Patel, A., Agarwal, Y.: Transfer learning for performance modeling of configurable systems: An exploratory analysis. In: Proc. of ASE'17, p. 497–508 (2017)
40. Jamshidi, P., Velez, M., Kästner, C., Siegmund, N.: Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In: Proc. of ESEC/FSE'18, p. 71–82 (2018). DOI 10.1145/3236024.3236074. URL `https://doi.org/10.1145/3236024.3236074`
41. Jamshidi, P., Velez, M., Kästner, C., Siegmund, N., Kawthekar, P.: Transfer learning for improving model predictions in highly configurable software. In: Proc. of SEAMS'17, pp. 31–41 (2017). DOI http://dx.doi.org/10.1109/SEAMS.2017.11. URL `https://arxiv.org/abs/1704.00234`
42. Johnson, S.C.: Hierarchical clustering schemes. Psychometrika **32**(3), 241–254 (1967)
43. Kaltenecker, C., Grebhahn, A., Siegmund, N., Apel, S.: The interplay of sampling and machine learning for software performance prediction. IEEE Softw. **37**(4), 58–66 (2020)
44. Kaltenecker, C., Grebhahn, A., Siegmund, N., Guo, J., Apel, S.: Distance-based sampling of software configuration spaces. In: Proc. of ICSE'19, p. 1084–1094 (2019). DOI 10.1109/ICSE.2019.00112. URL `https://doi.org/10.1109/ICSE.2019.00112`
45. Kendall, M.G.: Rank correlation methods. Griffin (1948)
46. Khavari Tavana, M., Sun, Y., Bohm Agostini, N., Kaeli, D.: Exploiting adaptive data compression to improve performance and energy-efficiency of compute workloads in multi-gpu systems. In: Proc. of IPDPS'19, pp. 664–674 (2019). DOI 10.1109/IPDPS.2019.00075

47. Kolesnikov, S., Siegmund, N., Kästner, C., Grebhahn, A., Apel, S.: Tradeoffs in modeling performance of highly configurable software systems. Software & Systems Modeling **18**(3), 2265–2283 (2018). DOI 10.1007/s10270-018-0662-9. URL `https://doi.org/10.1007/s10270-018-0662-9`
48. Kolesnikov, S., Siegmund, N., Kästner, C., Apel, S.: On the relation of external and internal feature interactions: A case study (2018)
49. Krishna, R., Nair, V., Jamshidi, P., Menzies, T.: Whence to learn? transferring knowledge in configurable systems using BEETLE. CoRR pp. 1–16 (2019). URL `http://arxiv.org/abs/1911.01817`
50. Larsson, H., Taghia, J., Moradi, F., Johnsson, A.: Source selection in transfer learning for improved service performance predictions. In: Proc. of Networking'21, pp. 1–9 (2021). DOI 10.23919/IFIPNetworking52078.2021.9472818
51. Leitner, P., Cito, J.: Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. ACM Trans. Internet Technol. **16**(3) (2016). DOI 10.1145/2885497. URL `https://doi.org/10.1145/2885497`
52. Lesoil, L., Acher, M., Blouin, A., Jézéquel, J.M.: Deep software variability: Towards handling cross-layer configuration. In: 15th International Working Conference on Variability Modelling of Software-Intensive Systems, VaMoS'21. Association for Computing Machinery, New York, NY, USA (2021). DOI 10.1145/3442391.3442402. URL `https://doi.org/10.1145/3442391.3442402`
53. Lillack, M., Müller, J., Eisenecker, U.W.: Improved prediction of non-functional properties in software product lines with domain context. Software Engineering 2013 **1**(1), 1–14 (2013)
54. Liu, F., Miniskar, N.R., Chakraborty, D., Vetter, J.S.: Deffe: A data-efficient framework for performance characterization in domain-specific computing. In: Proc. of CF'20, p. 182–191 (2020). DOI 10.1145/3387902.3392633. URL `https://doi.org/10.1145/3387902.3392633`
55. Maiorca, D., Biggio, B.: Digital investigation of pdf files: Unveiling traces of embedded malware. IEEE Security Privacy **17**(1), 63–71 (2019)
56. Massey Jr, F.J.: The kolmogorov-smirnov test for goodness of fit. Journal of the American statistical Association **46**(253), 68–78 (1951)
57. Maxiaguine, A., Yanhong Liu, Chakraborty, S., Wei Tsang Ooi: Identifying "representative" workloads in designing mpsoc platforms for media processing. In: Proc. of ESTImedia'04, pp. 41–46 (2004). URL `https://ieeexplore.ieee.org/document/1359702`
58. Molnar, C.: Interpretable Machine Learning. Lulu. com, Munich (2020)
59. Murwantara, I.M., Bordbar, B., Minku, L.L.: Measuring energy consumption for web service product configuration. In: Proc. of iiWAS'14, p. 224–228 (2014). DOI 10.1145/2684200.2684314. URL `https://doi.org/10.1145/2684200.2684314`
60. Mühlbauer, S., Apel, S., Siegmund, N.: Identifying software performance changes across variants and versions. In: Proc. of ASE'20, pp. 611–622 (2020)
61. Nair, V., Menzies, T., Siegmund, N., Apel, S.: Using bad learners to find good configurations. In: Proc. of ESEC/FSE'17, pp. 257–267 (2017)
62. Nair, V., Menzies, T., Siegmund, N., Apel, S.: Faster discovery of faster system configurations with spectral learning. Automated Software Engg. **25**(2), 247–277 (2018). DOI 10.1007/s10515-017-0225-2. URL `https://doi.org/10.1007/s10515-017-0225-2`
63. Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S.: Finding faster configurations using flash. IEEE Transactions on Software Engineering **46**(7), 794–811 (2020). DOI 10.1109/TSE.2018.2870895
64. Nelson, T.: Technically-oriented pdf collection (2014). URL `https://github.com/tpn/pdfs`
65. Oh, J., Batory, D., Myers, M., Siegmund, N.: Finding near-optimal configurations in product lines by random sampling. In: Proc. of ESEC/FSE'17, p. 61–71 (2017). DOI 10.1145/3106237.3106273. URL `https://doi.org/10.1145/3106237.3106273`
66. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. Commun. ACM **62**(11), 137–145 (2019). DOI 10.1145/3361566. URL `https://doi.org/10.1145/3361566`
67. Pereira, J.A., Acher, M., Martin, H., Jézéquel, J.M., Botterweck, G., Ventresque, A.: Learning software configuration spaces: A systematic literature review. JSS p.

111044 (2021). DOI https://doi.org/10.1016/j.jss.2021.111044. URL https://www.sciencedirect.com/science/article/pii/S0164121221001412
68. Plazar, Q., Acher, M., Perrouin, G., Devroey, X., Cordy, M.: Uniform sampling of sat solutions for configurable systems: Are we there yet? In: Proc. of ICST'19, pp. 240–251 (2019)
69. Plotnikov, D., Melnik, D., Vardanyan, M., Buchatskiy, R., Zhuykov, R., Lee, J.H.: Automatic tuning of compiler optimizations and analysis of their impact. Procedia Computer Science **18**, 1312–1321 (2013). DOI 10.1016/j.procs.2013.05.298. URL https://doi.org/10.1016/j.procs.2013.05.298
70. Poess, M., Floyd, C.: New tpc benchmarks for decision support and web commerce. SIGMOD Rec. **29**(4), 64–71 (2000). DOI 10.1145/369275.369291. URL https://doi.org/10.1145/369275.369291
71. Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S.: Performance analysis of private blockchain platforms in varying workloads. In: Proc. of ICCCN'17, pp. 1–7 (2017). DOI 10.1109/icccn.2017.8038517. URL https://doi.org/10.1109/icccn.2017.8038517
72. Porter, U.K.A.M.S.W.J.S.F.A.: Satune: A study-driven auto-tuning approach for configurable software verification tools (2021)
73. Pouchet, L.N., et al.: Polybench: The polyhedral benchmark suite. URL: http://www.cs. ucla. edu/pouchet/software/polybench **437**, 1–1 (2012)
74. Rapps, S., Weyuker, E.J.: Selecting software test data using data flow information. IEEE transactions on software engineering **SE-11**(4), 367–375 (1985)
75. Saleem, M.S., Ding, C., Liu, X., Chi, C.H.: Personalized decision-strategy based web service selection using a learning-to-rank algorithm. IEEE Transactions on Services Computing **8**(5), 727–739 (2015)
76. Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K.: Cost-efficient sampling for performance prediction of configurable systems. In: Proc. of ASE'30, p. 342–352 (2015). DOI 10.1109/ASE.2015.45. URL https://doi.org/10.1109/ASE.2015.45
77. Seber, G.A., Lee, A.J.: Linear regression analysis, vol. 329. John Wiley & Sons (2012)
78. Shu, Y., Sui, Y., Zhang, H., Xu, G.: Perf-al: Performance prediction for configurable software through adversarial learning. In: Proc. of ESEM'20 (2020). DOI 10.1145/3382494.3410677. URL https://doi.org/10.1145/3382494.3410677
79. Siegmund, N., Grebhahn, A., Apel, S., Kästner, C.: Performance-influence models for highly configurable systems. In: Proc. of ESEC/FSE'15, p. 284–294 (2015). DOI 10.1145/2786805.2786845. URL https://doi.org/10.1145/2786805.2786845
80. Siegmund, N., Kolesnikov, S.S., Kästner, C., Apel, S., Batory, D., Rosenmüller, M., Saake, G.: Predicting performance via automated feature-interaction detection. In: Proc. of ICSE'12, pp. 167–177 (2012)
81. Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G.: Spl conqueror: Toward optimization of non-functional properties in software product lines. Software Quality Journal **20**(3–4), 487–517 (2012). DOI 10.1007/s11219-011-9152-9. URL https://doi.org/10.1007/s11219-011-9152-9
82. Sinha, U., Cashman, M., Cohen, M.B.: Using a genetic algorithm to optimize configurations in a data-driven application. In: Proc. of SSBSE'20, pp. 137–152 (2020). DOI 10.1007/978-3-030-59762-7_10. URL https://doi.org/10.1007/978-3-030-59762-7_10
83. Still, M.: Imagemagick (2006)
84. Temple, P., Acher, M., Biggio, B., Jézéquel, J.M., Roli, F.: Towards adversarial configurations for software product lines (2018)
85. Temple, P., Acher, M., Jezequel, J.M., Barais, O.: Learning contextual-variability models. IEEE Software **34**(6), 64–70 (2017)
86. Temple, P., Acher, M., Jézéquel, J.M.A., Noel-Baron, L.A., Galindo, J.A.: Learning-based performance specialization of configurable systems. Research report, IRISA (2017). URL https://hal.archives-ouvertes.fr/hal-01467299
87. Temple, P., Galindo, J.A., Acher, M., Jézéquel, J.M.: Using machine learning to infer constraints for product lines. In: Proc. of SPLC'16 (2016)
88. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: Proc. of

KDD'13, p. 847–855 (2013). DOI 10.1145/2487575.2487629. URL https://doi.org/10.1145/2487575.2487629
89. Valov, P., Guo, J., Czarnecki, K.: Empirical comparison of regression methods for variability-aware performance prediction. In: Proc. of SPLC'15, p. 186–190 (2015)
90. Valov, P., Guo, J., Czarnecki, K.: Transferring pareto frontiers across heterogeneous hardware environments. In: Proc. of ICPE'20, p. 12–23 (2020). DOI 10.1145/3358960.3379127. URL https://doi.org/10.1145/3358960.3379127
91. Valov, P., Petkovich, J.C., Guo, J., Fischmeister, S., Czarnecki, K.: Transferring performance prediction models across different hardware platforms. In: Proc. of ICPE'17, p. 39–50 (2017)
92. Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B.: Automatic database management system tuning through large-scale machine learning. In: Proc. of ICMD'17, p. 1009–1024 (2017)
93. Velez, M., Jamshidi, P., Siegmund, N., Apel, S., Kästner, C.: White-box analysis over machine learning: Modeling performance of configurable systems (2021)
94. Wang, Y., Inguva, S., Adsumilli, B.: YouTube UGC dataset for video compression research. In: Proc. of MMSP'19), pp. 1–5 (2019). DOI 10.1109/mmsp.2019.8901772. URL https://doi.org/10.1109/mmsp.2019.8901772
95. Weber, M., Apel, S., Siegmund, N.: White-box performance-influence models: A profiling and learning approach (2021)
96. Weckesser, M., Kluge, R., Pfannemüller, M., Matthé, M., Schürr, A., Becker, C.: Optimal reconfiguration of dynamic software product lines based on performance-influence models. In: Proc. of SPLC'18 (2018)
97. Westermann, D., Happe, J., Krebs, R., Farahbod, R.: Automated inference of goal-oriented performance prediction functions. In: Proc. of ASE'12, p. 190–199 (2012). DOI 10.1145/2351676.2351703. URL https://doi.org/10.1145/2351676.2351703
98. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for sat. Journal of artificial intelligence research **32**, 565–606 (2008)
99. Zhang, Y., Guo, J., Blais, E., Czarnecki, K.: Performance prediction of configurable software systems by fourier learning (t). In: Proc. of ASE'15, pp. 365–373 (2015). DOI 10.1109/ASE.2015.15
100. Zhang, Y., Guo, J., Blais, E., Czarnecki, K., Yu, H.: A mathematical model of performance-relevant feature interactions. In: Proc. of SPLC'16, p. 25–34 (2016). DOI 10.1145/2934466.2934469. URL https://doi.org/10.1145/2934466.2934469
101. Zuluaga, M., Krause, A., Püschel, M.: e-pal: An active learning approach to the multi-objective optimization problem. Journal of Machine Learning Research **17**(104), 1–32 (2016)
102. Zwick, W.R., Velicer, W.F.: Factors influencing four rules for determining the number of components to retain. Multivariate Behavioral Research **17**(2), 253–269 (1982)
103. Švogor, I., Crnković, I., Vrček, N.: An extensible framework for software configuration optimization on heterogeneous computing systems: Time and energy case study. Information and Software Technology **105** (2019)