Sampling Effect on Performance Prediction of Configurable Systems: A Case Study

Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel

Sampling Effect on Performance Prediction of Configurable Systems: A Case Study

Juliana Alves Pereira
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
juliana.alves-pereira@irisa.fr

Mathieu Acher
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
mathieu.acher@irisa.fr

Hugo Martin
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
hugo.martin@irisa.fr

Jean-Marc Jézéquel
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
jean-marc.jezequel@irisa.fr

ABSTRACT

Numerous software systems are highly configurable and provide a myriad of configuration options that users can tune to fit their functional and performance requirements (*e.g.,* execution time). Measuring all configurations of a system is the most obvious way to understand the effect of options and their interactions, but is too costly or infeasible in practice. Numerous works thus propose to measure only a few configurations (a sample) to learn and predict the performance of any combination of options' values. A challenging issue is to sample a small and representative set of configurations that leads to a good accuracy of performance prediction models. A recent study devised a new algorithm, called distance-based sampling, that obtains state-of-the-art accurate performance predictions on different subject systems. In this paper, we replicate this study through an in-depth analysis of x264, a popular and configurable video encoder. We systematically measure all 1,152 configurations of x264 with 17 input videos and two quantitative properties (encoding time and encoding size). Our goal is to understand whether there is a dominant sampling strategy over the very same subject system (x264), *i.e.,* whatever the workload and targeted performance properties. The findings from this study show that random sampling leads to more accurate performance models. However, without considering random, there is no single "dominant" sampling, instead different strategies perform best on different inputs and non-functional properties, further challenging practitioners and researchers.

KEYWORDS

Software Product Lines, Configurable Systems, Machine Learning, Performance Prediction

1 INTRODUCTION

Configurable software systems offer a multitude of configuration options that can be combined to tailor the systems' functional behavior and performance (*e.g.,* execution time, memory consumption). Options often have a significant influence on performance properties that are hard to know and model *a priori*. There are numerous possible options values, logical constraints between options, and subtle interactions among options [15, 25, 46, 51, 52] that can have an effect while quantitative properties such as execution time are themselves challenging to comprehend.

Measuring all configurations of a configurable system is the most obvious path to *e.g.,* find a well-suited configuration, but is too costly or infeasible in practice. Machine-learning techniques address this issue by measuring only a subset of configurations (known as sample) and then using these configurations' measurements to build a performance model capable of predicting the performance of other configurations (*i.e.,* configurations not measured before). Several works thus follow a "sampling, measuring, learning" process [3, 15, 20–23, 27, 40, 42, 43, 46, 51, 52, 60, 63, 64]. A crucial step is the way the sampling is realized, since it can drastically affect the performance model accuracy [25, 46]. Ideally, the sample should be small to reduce the cost of measurements and representative of the configuration space to reduce prediction errors. The sampling phase involves a number of difficult activities: *(1)* picking configurations that are valid and conform to constraints among options – one needs to resolve a satisfiability problem; *(2)* instrumenting the executions and observations of software for a variety of configurations – it might have a high computational cost especially when measuring performance aspects of software; *(3)* guaranteeing a coverage of the configuration space to obtain a representative sample set. An ideal coverage includes all influential configuration options by covering different kinds of interactions relevant to performance. Otherwise, the learning may hardly generalize to the whole configuration space.

With the promise to select a small and representative sample set of valid configurations, several sampling strategies have been devised in the last years [46]. For example, random sampling aims to cover the configuration space uniformly while coverage-oriented sampling selects the sample set according to a coverage criterion (*e.g., t*-wise sampling to cover all combinations of t selected options). Recently, Kaltenecker *et al.* [25] analyzed 10 popular real-world software systems and found that their novel proposed sampling strategy, called diversified distance-based sampling, dominates five other sampling strategies by decreasing the cost of labelling software configurations while minimizing the prediction errors.

In this paper, we conduct a replication of this preliminary study by exclusively considering the x264 case, a configurable video encoder. Though we only consider one particular configurable system, we make vary its workloads (with the use of 17 input videos) and measured two performance properties (encoding time and encoding size) over 1,152 configurations. Interestingly, Kaltenecker *et al.* [25] also analyzed the same 1,152 configurations of x264, but a fixed

input video was used and only the time was considered. The goal of our experiments is to determine whether sampling strategies considered in [25] over different subject systems are as effective on the *same* configurable system, but with different factors possibly influencing the distribution of the configuration space. A hypothesis is that practitioners of a configurable system can rely on a one-size-fits-all sampling strategy that is cost-effective whatever the factors influencing the distribution of its configuration space. On the contrary, another hypothesis is that practitioners should change their sampling strategy each time an input (here: videos) or a performance property are targeted.

We investigate to what extent sampling strategies are sensitive to different workloads of the x264 configurable system and different performance properties: What are the most effective sampling strategies? Is random sampling a strong baseline? Is there a dominant sampling strategy that practitioners can always rely on? To this end, we systematically report the influence of sampling strategies on the accuracy of performance predictions and on the robustness of prediction accuracy.

Our contributions are as follows:

- We rank six sampling strategies based on a unified benchmark over seventeen different inputs and two non-functional properties. We show that the ranking of non-random sampling strategies is quite unstable, being heavily sensitive to input video and targeted performance property. We gather preliminary insights about the effectiveness of some sampling strategies on some videos and performances;
- We compare our results to the previous study of x264 [25], that was based on the use of a single input video and measurements of encoding time. Through our experimental results, we find that for the non-functional property encoding time uniform random sampling dominates all the others sampling strategies w.r.t. prediction accuracy for a majority of cases, as in Kaltenecker et al. [25];
- We have made all the artifacts from our replication study publicly available at https://github.com/jualvespereira/ICPE2020.

With respect to the categorized research methods by Stol et al. [57], our paper mainly contributes to a knowledge-seeking study. Specifically, we perform a field experiment of x264, a highly-configurable system and mature project. We gain insights about performance' properties using a large corpus of configurations and input videos.

Audience. Researchers and practitioners in configurable systems and performance engineering shall benefit from our sampling experiments and insights. We discuss impacts of our results for practitioners (How to choose a sampling strategy?) and for the research community (*e.g.,* on the design and assessment of sampling strategies).

2 BACKGROUND AND RELATED WORK

In this section, we introduce basic concepts of configurable software systems and motivate the use of learning techniques in this field. Furthermore, we briefly describe six state-of-the-art sampling strategies used in our experiments.

2.1 Learning Software Configuration Spaces

x264 is a command-line tool to encode video streams into the H.264/MPEG-4 AVC format. Users can configure x264 through the selection of numerous options, some having an effect on the time needed to encode a video, on the quality or the size of the output video, *etc.* A configuration of x264 is an assignment of values to options. In our study and as in [25], we only consider Boolean options that can be selected or deselected. As in most configurable systems, not all combinations of options are valid due to constraints among options. For instance, `ref_1`, `ref_5`, and `ref_9` are mutually exclusive and at least one of these options should be selected.

Executing and measuring every valid configuration to know about its performance or identify the performance-optimal one is often unfeasible or costly. To overcome this problem, machine learning techniques rely on a small and representative *sample* of configurations (see Figure 1). Each configuration of the sample is executed and labelled with a performance measurement (*e.g.,* encoding time). The sample is then used for *training* a learning algorithm (*i.e.,* a regressor) that builds a performance model capable of predicting the performance of unmeasured configurations. The performance model may lead to prediction errors. The overall goal is to obtain high *accuracy* roughly computed as the difference between actual performances and predicted performances (more details are given hereafter). How to efficiently sample, measure, and learn is subject to intensive research [46]. In case important (interactions among) options are not included in the training set, the learning phase can hardly generalize to the whole population of configurations. Hence, sampling is a crucial step of the overall learning process with an effect on the accuracy of the prediction model.

2.2 Sampling Strategies

Several sampling strategies have been proposed in the literature about software product lines and configurable systems [46, 62, 66].

Sampling for testing. Some works consider sampling for the specific case of testing configurations. There is not necessarily a learning phase and the goal of sampling is mostly to find and cover as many faults as possible. For instance, Medeiros *et al.* compared 10 sampling algorithms to detect different faults in configurable systems [35]. Arcuri *et al.* theoretically demonstrate that a uniform random sampling strategy may outperform coverage-based sampling [5] (see hereafter). Halin *et al.* demonstrated that uniform random sampling forms a strong baseline for faults and failure efficiency on the JHipster case [17]. Varshosaz *et al.* [66] conducted a survey of sampling for testing configurable systems. Though the purpose differs, some of these sampling strategies are also relevant and considered in the context of performance prediction.

Sampling for learning. Pereira *et al.* [46] review several sampling strategies specifically used for learning configuration spaces. We now present an overview of six sampling strategies also considered in [25] and used in our study. All strategies have the merit of being agnostic of the domain (no specific knowledge or prior analysis are needed) and are directly applicable to any configurable system.

Random sampling aims to cover the configuration space *uniformly*. Throughout the paper, we refer to random as uniform random sampling. The challenge is to select one configuration amongst all the *valid* ones in such a way each configuration receives an equal probability to be included in the sample. An obvious solution is to enumerate all valid configurations .and randomly pick a sample from the whole population. However, enumerative approaches

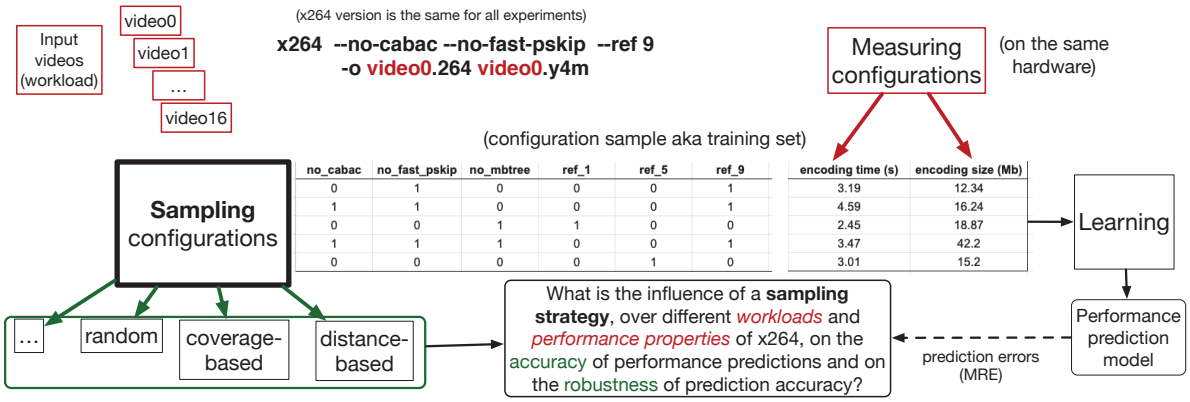


Figure 1: Design study: sampling effect on performance predictions of x264 configurations

quickly do not scale with a large number of configurations. Oh *et al.* [43] rely on binary decision diagrams to compactly represent a configuration space, which may not scale for very large systems [36]. Another line of research is to rely on satisfiability (SAT) solvers. For instance, UniGen [8, 9] uses a hashing-based functions to synthesize samples in a nearly uniform manner with strong theoretical guarantees. These theoretical properties come at a cost: the hashing-based approach requires adding large clauses to formulas. Plazar *et al.* [47] showed that state-of-the-art algorithms are either not able to produce any sample or unable to generate uniform samples for the SAT instances considered. Overall, a true uniform random sampling may be hard to realize, especially for large configurable systems. At the scale of the x264 study [25], though, uniform sampling is possible (the whole population is 1,152 configurations). The specific question we explore here is whether random is effective for learning (in case it is applicable as in x264).

When random sampling is not applicable, several alternate techniques have been proposed typically by sacrificing some uniformity for a substantial increase in performance.

Solver-based. Many works rely on off-the-shelf constraint solver, such as SAT4J [31] or Z3 [12], for sampling. For instance, a random seed can be set to the Z3 solver and internally influences the variable selection heuristics, which can have an effect on the exploration of valid configurations. Henard *et al.* noticed that solvers' internal order yields non-uniform (and predictable) exploration of the configuration space [18]. Hence, these strategies do not guarantee true randomness as in uniform random sampling. Often the sample set consists of a locally clustered set of configurations.

Randomized solver-based. To weaken the locality drawback of solver-based sampling, Henard *et al.* change the order of variables and constraints at each solver run. This strategy, called *randomized solver-based sampling* in Kaltenecker et al. [25], increases diversity of configurations. Though it cannot give any guarantees about randomness, the diversity may help to capture important interactions between options for performance prediction.

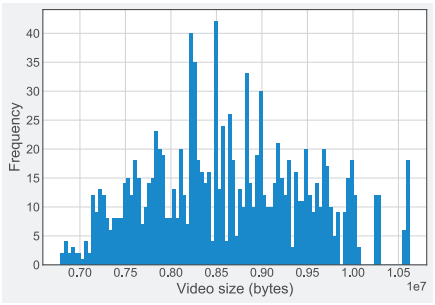
Coverage-based sampling aims to optimize the sample with regards to a coverage criterion. Many criteria can be considered such as statement coverage that requires the analysis of the source code. In this paper and as in Kaltenecker et al. [25], we rely on t-wise sampling [11, 24, 30]. This sampling strategy selects configurations to cover all combinations of t selected options. For instance, pair-wise (t=2) sampling covers all pairwise combinations of options being selected. There are different methods to compute t-wise sampling. As in [25], we rely on the implementation of Siegmund *et al.* [53].

Distance-based. Kaltenecker *et al.* [25] propose distance-based sampling. The idea is to cover the configuration space by selecting configurations according to a given probability distribution (typically a uniform distribution) and a distance metric. The underlying benefit is that distance-based sampling can better scale compared to an enumerative-based random sampling, while the generated samples are closed to those obtained with a uniform random.

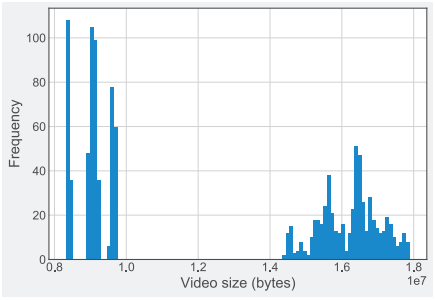
Diversified distance-based sampling is a variant of distance-based sampling [25]. The principle is to increase diversity of the sample set by iteratively adding configurations that contain the least frequently selected options. The intended benefit is to avoid missing the inclusion of some (important) options in the process.

2.3 Sampling Effect on x264

This paper aims to replicate the study of Kaltenecker *et al.* [25]. While Kaltenecker *et al.* analyse a wide variety of systems from different domains, we focus on the analysis of a single configurable system. Compared to that paper, our study enables a deeper analysis of sampling strategies over the possible influences of inputs and performance properties. Specifically, we analyze a set of seventeen input videos and two non-functional properties. As in [25], we collect results of 100 independent experiments to increase statistical confidence and external validity. We aim at exploring whether the results obtained in [25] may be generalized over different variations of a single configurable system. Interestingly, numerous papers have specifically considered the x264 configurable system for assessing their proposals [15, 21, 22, 43, 46, 51, 58, 63, 64], but a fixed performance property or input video is usually considered. Jamshidi *et al.* [21] explore the impacts of versions, workloads, and hardware of several configurable systems, including x264. However, the effect of sampling strategies was not considered. In [22], sampling



(a) flower_sif.y4m $x264_2$



(b) 720p50_parkrun_ter.y4m $x264_{15}$

Figure 2: The size distribution of 1,152 configurations of x264 over two input videos

strategies for transfer learning were investigated; we are not considering such scenarios in our study. Overall, given a configurable system like x264, practitioners face the problem of choosing the right techniques for "sampling, measuring, learning". Specifically, we aim to understand what sampling strategy to use and whether there exists a sampling to rule any configuration space of x264.

The use of different input videos obviously changes the raw and absolute performance values, but it can also change the overall distribution of configuration measurements. Figure 2 gives two distributions over two input videos for the performance property *size* for the whole population of valid configurations. Pearson correlation between the performance measurements of $x264_2$ and $x264_{15}$ is -0.35, suggesting a weak, negative correlation. The differences among distributions question the existence of a one-size-fits-all sampling capable of generating a representative sample set whatever the input videos or performance properties. Another hypothesis is that the way the sampling is done can pay off for some distributions but not for all.

Given the vast variety of possible input videos and performance properties that may be considered, the performance variability of configurations grows even more. Our aim is to investigate how such factors affect the overall prediction accuracy of different sampling strategies: *Is there a dominant sampling strategy for performance prediction of the same configurable system?*

3 DESIGN STUDY

In this section, we introduce our research questions, the considered subject system, and the experiment setup.

3.1 Research Questions

We conducted a series of experiments to evaluate six sampling strategies and to compare our results to the original results in [25]. We aim at answering the following two research questions:

- **(RQ1)** What is the influence of using different sampling strategies on the accuracy of performance predictions over different inputs and non-functional properties?
- **(RQ2)** What is the influence of randomness of using different sampling strategies on the robustness of prediction accuracy?

It is not new the claim that the prediction accuracy of machine learning extensively depends on the sampling strategy. The originality of the research question is to what extent are performance prediction models of the *same* configurable system (here: x264) sensitive to other factors, such as different inputs and non-functional properties. To address *RQ1*, we analyze the sensitivity of the prediction accuracy of sampling strategies to these factors. Since most of the considered sampling strategies use randomness, which may considerably affect the prediction accuracy, *RQ2* quantitatively compares whether the variances (over 100 runs) on prediction accuracy between different sampling strategies and sample sizes differ significantly. We show that the sampling prediction accuracy and robustness hardly depends on the definition of performance (*i.e.*, encoding time or encoding size). As in Kaltenecker et al. [25], we have excluded t-wise sampling from *RQ2*, as it is also deterministic in our setting and does not lead to variations.

3.2 Subject System

We conduct an in-depth study of x264, a popular and highly configurable video encoder implemented in C. We choose x264 instead of the other case studies documented in Kaltenecker et al. [25] because x264 demonstrated more promising accuracy results to the newest proposed sampling approach (*i.e.*, diversified distance-based sampling). With this study, we aim at investigating, for instance, whether diversified distance-based sampling also dominates across different variations of x264 (*i.e.*, inputs, performance properties). As benchmark, we encoded 17 different input videos from raw YUV to the H.264 codec and measured two quantitative properties (encoding time and encoding size).

- Encoding time (in short **time**): how many seconds x264 takes to encode a video.
- Encoding size of the output video (in short **size**): compression size (in bytes) of an output video in the H.264 format.

All measurements have been performed over the same version of x264 and on a grid computing infrastructure called IGRIDA[1]. Importantly, we used the same hardware characteristics for all performance measurements. In Table 1, we provide an overview of the encoded input videos. This number of inputs allows us to draw conclusions about the practicality of sampling strategies for diversified conditions of x264.

To control measurement bias while measuring execution time, we have repeated the measurements several times. We report in Table 1 how many times we have repeated the time measurements of all 1,152 configurations for each video input. the number of

[1] http://igrida.gforge.inria.fr/

	video	#times	stability
$x264_0$	bridge_far_cif.y4m	5	0.010127
$x264_1$	ice_cif.y4m	5	0.044476
$x264_2$	flower_sif.y4m	5	0.036826
$x264_3$	claire_qcif.y4m	5	0.086958
$x264_4$	sintel_trailer_2k_480p24.y4m	9	0.009481
$x264_5$	football_cif.y4m	5	0.029640
$x264_6$	crowd_run_1080p50.y4m	3	0.005503
$x264_7$	blue_sky_1080p25.y4m	8	0.010468
$x264_8$	FourPeople_1280x720_60.y4m	11	0.011258
$x264_9$	sunflower_1080p25.y4m	4	0.006066
$x264_{10}$	deadline_cif.y4m	23	0.014536
$x264_{11}$	bridge_close_cif.y4m	5	0.009892
$x264_{12}$	husky_cif.y4m	5	0.028564
$x264_{13}$	tennis_sif.y4m	5	0.044731
$x264_{14}$	riverbed_1080p25.y4m	3	0.007625
$x264_{15}$	720p50_parkrun_ter.y4m	8	0.010531
$x264_{16}$	soccer_4cif.y4m	16	0.011847

Table 1: Overview of encoded input videos including the number of times we measured the encoding time in order to ensure we have a stable set of measurements (according to RSD results), independent of the machine.

repetitions has been increased to reach a standard deviation of less than 10%. Since measurements are costly, we set the repetition to up 30 times given 1-hour restriction from where we got the $\#times$ in Table 1. We repeated the measurements at least three times and at most 23 times and retained the average execution time for each configuration. The *stability* column reports whether the configuration measurements for a given input video are stable (Relative Standard Deviation - RSD). For example, measurements have been repeated 5 times for video 0 (bridge_far_cif.y4m) and the measurements present low RSD of $\approx$ 1%. Although the RSD measure for Video 3 (claire_qcif.y4m) is higher compared to the others, the deviation still remains lower than 10% (*i.e.,* RSD $\approx$ 8.7%). We provide the variability model and the measurements of each video input for encoding time and encoding size on our supplementary website.

3.3 Experiment Setup

In our experiments, the independent variables are the choice of the input videos, the predicted non-functional-property, the sample strategies and the sample sizes.

For comparison, we used the same experiment design than in Kaltenecker et al. [25]. To evaluate the accuracy of different sampling strategies over different inputs and non-functional-properties, we conducted experiments using three different sample sizes. To be able to use the same sample sizes for all sampling strategies, we consider the sizes from t-wise sampling with t=1, t=2, and t=3. As described in Section 2.2, t-wise sampling covers all combinations of t configuration options being selected. We learn a performance model based on the sample sets along with the corresponding performance measurements defined by the different sampling strategies. Although several machine-learning techniques have been proposed in the literature with this purpose [46], such as linear regression [10, 20, 21, 25–27, 33, 39, 50, 52, 54, 67], classification and regression trees (CART) [2, 16, 21, 28, 39–42, 48, 51, 56, 58, 60, 63, 64, 68–71], and random forest [3, 6, 49, 61, 63]. In this paper, we use stepwise multiple linear regression [52] as in Kaltenecker et al. [25]. According to Kaltenecker et al. [25], multiple linear regression is often as accurate as CART and random forests.

To calculate the prediction error rate, we use the resulting performance models to predict the performance of the entire dataset of valid configurations C. We calculate the error rate of a prediction model in terms of the *mean relative error* (MRE - Equation 1). MRE is used to estimate the accuracy between the exact measurements and the predicted one.

$$MRE = \frac{1}{|C|} \sum_{c \in C} \frac{|measured_c - predicted_c|}{measured_c} \quad (1)$$

Where C is the set of all valid configurations used as the validation set, and $measured_c$ and $predicted_c$ indicate the measured and predicted values of performance for configuration c with $c \in C$, respectively. The exact value of $measured_c$ is measured at runtime while running the configuration c, and the predicted values of $predicted_c$ is computed based on the model built with a sample of configurations t (see Section 2.2). To address *RQ1*, we computed the mean error rate for each input video and sample size. A lower error rate indicates a higher accuracy. Then, we use a Kruskal-Wallis test [29] and pair-wise one-sided Mann-Whitney U tests [34] to identify whether the error rate of two sampling strategies differs significantly ($p < 0.05$) [4]. In addition, we compute the effect size $\hat{A}_{12}$ [65] (small(>0.56), medium(>0.64), and large(>0.71)) to easily compare the error rates of two sampling strategies.

To address *RQ2*, we compute the variance across the error rates over 100 runs. A lower variance indicates higher robustness. First, we use Levene's test [32] to identify whether the variances of two sampling strategies differ significantly from each other. Then, for these sampling strategies, we perform a one-sided F-tests [55] to compare pair-wisely the variance between sampling strategies.

All sampling and learning experiments have been performed on the same machine with Intel Core i7 CPU 2,2 GHz and 4GB RAM. To reduce fluctuations in the values of dependent variables caused by randomness (*e.g.,* the random generation of input samples), we evaluated each combination of the independent variables 100 times. That is, for each input video, non-functional property, sampling strategy and sampling size, we instantiated our experimental settings and measured the values of all dependent variables 100 times with random seeds from 1 to 100.

4 RESULTS

We compare six sampling strategies: t-wise, solver-based, randomized solver-based, distance-based, diversified distance-based, and random. Next, we present the results regarding prediction accuracy (*RQ1*, Section 4.1) and robustness (*RQ2*, Section 4.2).

Video	Coverage-based			Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
$x264_0$	18.2 %	13.9 %	13.4 %	24.0 %	27.0 %	27.5 %	22.3 %	19.9 %	24.3 %	16.5 %	12.7 %	10.6 %	16.3 %	8.8 %	**8.2 %**	16.7 %	9.2 %	8.2 %
$x264_1$	15.4 %	13.2 %	12.1 %	26.9 %	23.7 %	24.9 %	21.4 %	21.5 %	23.2 %	17.3 %	14.2 %	9.5 %	17.4 %	9.8 %	**8.7 %**	16.1 %	**9.2 %**	8.7 %
$x264_2$	29.3 %	10.3 %	9.7 %	21.4 %	19.4 %	16.4 %	19.1 %	19.6 %	19.4 %	17.4 %	11.4 %	9.8 %	17.6 %	9.6 %	9.3 %	**15.3 %**	9.5 %	9.3 %
$x264_3$	21.4 %	13.7 %	10.1 %	25.2 %	25.3 %	26.4 %	16.4 %	22.3 %	24.8 %	13.6 %	10.7 %	10.2 %	**12.8 %**	**9.8 %**	9.7 %	14.5 %	9.8 %	**9.2 %**
$x264_4$	21.8 %	12.3 %	14.4 %	23.9 %	21.2 %	22.0 %	18.3 %	21.1 %	22.5 %	14.2 %	11.7 %	9.7 %	13.9 %	10.1 %	8.9 %	13.9 %	**9.4 %**	**8.8 %**
$x264_5$	26.1 %	14.1 %	13.2 %	28.8 %	23.2 %	24.1 %	21.8 %	22.5 %	23.3 %	16.4 %	13.4 %	11.4 %	16.8 %	10.7 %	9.5 %	**15.7 %**	**10.0 %**	**9.3 %**
$x264_6$	25.9 %	18.1 %	8.6 %	23.6 %	28.5 %	29.1 %	18.2 %	21.6 %	24.9 %	13.7 %	9.9 %	9.0 %	13.2 %	8.8 %	7.8 %	**12.6 %**	**8.0 %**	**7.3 %**
$x264_7$	23.3 %	14.2 %	12.0 %	20.2 %	25.3 %	26.1 %	15.3 %	23.0 %	23.8 %	12.2 %	9.2 %	7.2 %	10.8 %	8.5 %	**7.2 %**	11.4 %	8.2 %	7.3 %
$x264_8$	20.8 %	13.1 %	11.5 %	20.3 %	22.7 %	23.6 %	16.7 %	23.4 %	23.4 %	12.6 %	10.4 %	9.6 %	11.1 %	9.3 %	8.3 %	12.0 %	8.7 %	**7.6 %**
$x264_9$	23.4 %	13.2 %	5.6 %	22.1 %	28.6 %	29.7 %	16.8 %	24.2 %	25.3 %	11.4 %	6.5 %	6.5 %	9.2 %	**5.8 %**	5.4 %	10.9 %	6.6 %	**5.4 %**
$x264_{10}$	21.9 %	12.3 %	9.3 %	22.6 %	23.2 %	24.0 %	17.9 %	22.4 %	24.3 %	14.0 %	10.2 %	9.7 %	13.5 %	9.4 %	8.9 %	14.0 %	**9.0 %**	**8.8 %**
$x264_{11}$	21.1 %	12.6 %	10.3 %	25.7 %	23.5 %	23.8 %	20.0 %	21.1 %	24.7 %	13.3 %	10.8 %	10.4 %	13.0 %	10.1 %	9.7 %	13.9 %	**9.4 %**	**9.1 %**
$x264_{12}$	25.4 %	13.4 %	10.4 %	26.2 %	21.2 %	21.6 %	19.8 %	20.6 %	20.9 %	16.2 %	13.7 %	10.9 %	16.3 %	11.4 %	9.1 %	**15.0 %**	**9.7 %**	**8.5 %**
$x264_{13}$	16.4 %	10.5 %	10.0 %	20.6 %	18.8 %	19.1 %	18.3 %	19.4 %	19.8 %	16.0 %	13.9 %	10.0 %	16.2 %	10.5 %	9.6 %	**15.5 %**	**9.7 %**	**9.0 %**
$x264_{14}$	20.7 %	16.9 %	15.8 %	34.3 %	39.5 %	40.6 %	28.5 %	29.7 %	32.4 %	18.1 %	11.1 %	9.6 %	18.4 %	7.8 %	7.3 %	17.4 %	**7.5 %**	**7.2 %**
$x264_{15}$	26.2 %	12.7 %	11.1 %	23.2 %	26.5 %	27.2 %	20.3 %	22.7 %	25.1 %	15.1 %	11.9 %	10.7 %	14.8 %	10.6 %	9.5 %	**13.9 %**	**9.1 %**	**8.9 %**
$x264_{16}$	22.9 %	12.3 %	8.4 %	22.1 %	24.5 %	25.2 %	18.0 %	22.2 %	23.6 %	13.4 %	9.4 %	8.9 %	12.6 %	8.5 %	7.8 %	12.5 %	**8.1 %**	**7.4 %**
Mean	22.4 %	13.3 %	10.9 %	24.2 %	24.8 %	25.4 %	19.4 %	22.2 %	23.9 %	14.8 %	11.3 %	9.6 %	14.3 %	9.4 %	8.5 %	**14.2 %**	**8.9 %**	**8.2 %**

Table 2: Error rates of t-wise, (randomized) solver-based, (diversified) distance-based, and random sampling for the prediction of encoding time for 17 input videos of x264. The bottom row contains the MRE mean across all input videos. The best results per input video and sample size are highlighted in bold if the Mann-Whitney U test reported a significant difference ($p < 0.05$).

Mann-Whitney U test [p value ($\hat{A}_{12}$)]																		
	Coverage-based			Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
Coverage-based					0 (0.99)	0 (1.00)		0 (0.98)	0 (1.00)									
Solver-based	10^{-08} (0.55)								10^{-49} (0.65)									
Randomized solver-based	10^{-120} (0.73)			10^{-99} (0.71)														
Distance-based	0 (0.92)	10^{-119} (0.73)	10^{-115} (0.73)	0 (0.92)	0 (0.99)	0 (1.00)	10^{-248} (0.83)	0 (0.98)	0 (1.00)									
Diversified distance-based	0 (0.93)	0 (0.95)	0 (0.94)	0 (0.93)	0 (1.00)	0 (1.00)	10^{-283} (0.86)	0 (1.00)	0 (1.00)	10^{-05} (0.54)	10^{-166} (0.77)	10^{-109} (0.72)						
Random	0 (0.93)	0 (0.97)	0 (0.96)	0 (0.93)	0 (1.00)	0 (1.00)	10^{-285} (0.86)	0 (1.00)	0 (1.00)	10^{-11} (0.57)	10^{-262} (0.84)	10^{-161} (0.77)	10^{-03} (0.53)	10^{-42} (0.63)	10^{-24} (0.60)			

Table 3: p values from a one-sided pair-wise Mann-Whitney U test for the property encoding time, where we tested pair-wisely whether the error rate of the sampling strategy from the row is significantly lower than the error rate of the sampling strategy from the column, for different sample sizes. The effect size is included for every significant result ($p < 0.05$), where we consider the effect as small, medium, and large when $\hat{A}_{12}$ is over 0.56, 0.64, and 0.71, respectively.

4.1 Results RQ1—Prediction Accuracy

In Tables 2 and 4, we show the MRE for the different sampling strategies and sample sizes for both encoding time and encoding size, respectively. In the bottom row, we provide the MRE mean over all input videos. As in Kaltenecker et al. [25], for each input video and sample-set size we highlight the lowest, statistically significant MRE in bold.

4.1.1 Input Sensitivity: Encoding Time. Random sampling performs best to all other sampling strategies or similar to the best one for t=1, t=2 and t=3 (except for a few input videos – $x264_1$, $x264_3$, $x264_7$, $x264_8$, $x264_9$, $x264_{10}$, $x264_{11}$ for t=1). We observe that for t=1 diversified distance-based sampling outperforms random sampling for seven input videos ($x264_0$, $x264_3$, $x264_7$, $x264_8$, $x264_9$, $x264_{10}$, $x264_{11}$). Overall, diversified distance-based sampling produces partially good results (close to random sampling). Diversified distance-based sampling outperforms the pure distance based sampling, however their error rates are very similar for t=1.

Solver-based sampling results in inaccurate performance models for all input videos and sample-set sizes. t-wise sampling performs overall better than solver-based sampling; randomized solver-based sampling performs best when only a very limited number of samples are considered (*i.e.,* t=1).

Table 3 reports the p value ($\hat{A}_{12}$) of the Mann-Whitney U test. This table shows whether the sampling strategy of the row has a significantly lower error rate than the sampling strategy of the column. To this end, we first performed Kruskal-Wallis tests for all sample sizes (t=1, t=2, and t=3). Then, whether we identified p values less than 0.05, indicating that, at least, two strategies differ significantly for each sample size [25], we performed one-sided Mann-Whitney U tests pair-wisely and, if significant ($p < 0.05$), we report the effect sizes in Table 3.

The first row shows that t-wise sampling has significantly lower error rates than solver-based sampling and randomized solver-based sampling for t=2 and t=3, with large effect sizes. In the second and third rows, we observe that solver-based sampling performs

Video	Coverage-based			Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
$x264_0$	**12.3 %**	11.6 %	11.1 %	12.3 %	11.4 %	11.3 %	25.1 %	12.7 %	13.3 %	25.3 %	12.5 %	10.6 %	23.3 %	10.6 %	9.2 %	13.1 %	**9.8 %**	**9.1 %**
$x264_1$	4.0 %	3.9 %	3.8 %	3.1 %	3.8 %	3.8 %	**1.7 %**	**3.8 %**	**3.8 %**	4.0 %	4.0 %	3.8 %	3.9 %	3.8 %	3.8 %	3.9 %	3.8 %	3.8 %
$x264_2$	14.9 %	14.3 %	4.8 %	**5.1 %**	4.7 %	4.7 %	15.9 %	4.7 %	**4.6 %**	14.3 %	14.0 %	10.2 %	13.8 %	12.0 %	4.7 %	7.6 %	**4.7 %**	4.6 %
$x264_3$	8.6 %	8.3 %	7.8 %	8.1 %	**7.3 %**	7.4 %	11.2 %	7.6 %	7.4 %	9.9 %	9.3 %	8.0 %	9.6 %	8.3 %	7.5 %	7.7 %	7.4 %	**7.3 %**
$x264_4$	18.4 %	16.7 %	**6.6 %**	**4.5 %**	6.8 %	6.8 %	14.1 %	**6.7 %**	6.7 %	17.5 %	16.7 %	7.0 %	16.9 %	6.9 %	6.9 %	7.8 %	6.9 %	6.9 %
$x264_5$	11.3 %	11.0 %	10.8 %	**4.9 %**	6.6 %	5.7 %	12.3 %	9.4 %	**4.8 %**	11.8 %	11.5 %	10.9 %	11.6 %	10.6 %	10.0 %	9.4 %	6.4 %	5.2 %
$x264_6$	24.6 %	**5.3 %**	**5.2 %**	**5.4 %**	5.4 %	5.3 %	25.6 %	5.3 %	5.3 %	17.6 %	16.8 %	5.5 %	16.1 %	5.4 %	5.4 %	6.3 %	5.3 %	5.3 %
$x264_7$	9.4 %	9.0 %	8.7 %	**8.1 %**	8.4 %	8.3 %	8.4 %	**8.2 %**	**8.2 %**	9.4 %	9.4 %	8.9 %	9.3 %	8.6 %	8.5 %	9.1 %	8.4 %	8.3 %
$x264_8$	10.4 %	9.7 %	8.9 %	8.7 %	8.0 %	8.1 %	11.2 %	**7.6 %**	**8.0 %**	12.4 %	12.0 %	9.5 %	12.0 %	9.9 %	8.5 %	**8.5 %**	8.3 %	8.2 %
$x264_9$	11.6 %	10.5 %	9.5 %	7.6 %	8.6 %	8.5 %	**6.9 %**	**8.4 %**	**8.4 %**	11.3 %	11.6 %	9.6 %	10.8 %	9.7 %	8.7 %	8.8 %	8.5 %	8.4 %
$x264_{10}$	5.2 %	5.2 %	4.9 %	5.2 %	5.0 %	4.8 %	5.0 %	4.6 %	4.6 %	6.0 %	5.8 %	5.0 %	5.7 %	5.1 %	4.7 %	**4.9 %**	4.6 %	4.6 %
$x264_{11}$	12.4 %	11.8 %	11.1 %	11.1 %	10.8 %	11.0 %	**8.8 %**	9.9 %	11.4 %	12.8 %	11.8 %	9.0 %	12.0 %	10.2 %	**8.6 %**	10.9 %	**9.4 %**	8.8 %
$x264_{12}$	25.7 %	3.6 %	3.6 %	5.3 %	3.5 %	3.6 %	28.9 %	3.6 %	3.5 %	16.5 %	14.6 %	3.5 %	15.4 %	3.5 %	3.4 %	**4.8 %**	**3.5 %**	**3.4 %**
$x264_{13}$	4.7 %	4.7 %	4.6 %	**4.5 %**	4.7 %	4.7 %	5.4 %	4.8 %	4.7 %	5.1 %	5.0 %	4.8 %	5.0 %	4.7 %	4.7 %	5.0 %	**4.7 %**	**4.6 %**
$x264_{14}$	10.2 %	9.6 %	9.4 %	5.1 %	**7.4 %**	**8.8 %**	**3.6 %**	9.6 %	9.5 %	10.6 %	10.6 %	10.0 %	9.8 %	9.6 %	9.6 %	9.3 %	9.0 %	9.5 %
$x264_{15}$	**4.1 %**	**4.0 %**	**4.0 %**	7.5 %	4.5 %	4.3 %	40.9 %	4.3 %	4.2 %	21.7 %	8.3 %	4.1 %	19.1 %	4.1 %	4.1 %	5.4 %	4.2 %	4.1 %
$x264_{16}$	8.3 %	8.1 %	7.9 %	**7.7 %**	7.8 %	7.6 %	9.2 %	7.7 %	7.6 %	8.8 %	8.7 %	8.2 %	8.7 %	7.9 %	7.7 %	8.3 %	**7.7 %**	**7.6 %**
Mean	11.5 %	8.7 %	7.2 %	**6.7 %**	6.8 %	6.7 %	13.8 %	7.0 %	6.8 %	12.6 %	10.7 %	7.6 %	12.0 %	7.7 %	6.8 %	7.7 %	6.6 %	**6.5 %**

Table 4: Error rates of t-wise, (randomized) solver-based, (diversified) distance-based, and random sampling for the prediction of encoding size for 17 input videos of x264. The bottom row contains the MRE mean across all input videos. The best results per input video and sample size are highlighted in bold if the Mann-Whitney U test reported a significant difference ($p < 0.05$).

Mann-Whitney U test [p value ($\hat{A}_{12}$)]																		
	Coverage-based			Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
Coverage-based							0.01 (0.52)			10^{-33} (0.62)	10^{-176} (0.78)	10^{-78} (0.68)	10^{-08} (0.55)					
Solver-based	10^{-232} (0.82)	10^{-268} (0.85)	10^{-43} (0.64)				10^{-152} (0.76)			10^{-247} (0.83)	0 (0.92)	10^{-145} (0.75)	10^{-219} (0.81)	10^{-86} (0.69)		10^{-95} (0.70)		
Randomized solver-based		10^{-272} (0.85)	10^{-61} (0.66)			10^{-05} (0.54)				10^{-13} (0.57)	0 (0.93)	10^{-184} (0.79)	0.01 (0.52)	10^{-94} (0.70)	10^{-06} (0.54)			
Distance-based																		
Diversified distance-based		10^{-76} (0.68)	10^{-70} (0.68)							10^{-10} (0.56)	10^{-293} (0.86)	10^{-191} (0.79)						
Random	10^{-180} (0.78)	0 (0.92)	10^{-183} (0.79)		10^{-28} (0.61)	10^{-58} (0.66)	10^{-83} (0.69)	10^{-22} (0.60)	10^{-37} (0.63)	10^{-228} (0.82)	0 (0.96)	0 (0.88)	10^{-179} (0.78)	10^{-193} (0.79)	10^{-71} (0.68)			

Table 5: p values from a one-sided pair-wise Mann-Whitney U test for the property encoding size, where we tested pair-wisely whether the error rate of the sampling strategy from the row is significantly lower than the error rate of the sampling strategy from the column, for different sample sizes. The effect size is included for every significant result ($p < 0.05$), where we consider the effect as small, medium, and large when $\hat{A}_{12}$ is over 0.56, 0.64, and 0.71, respectively.

significantly better than randomized solver-based sampling for t=3 with medium effect sizes; and randomized solver-based leads to lower error rates than t-wise sampling for t=1, with a large effect size. In the last three rows, we see that (diversified) distance-based sampling and random sampling lead to lower error rates than t-wise sampling, solver-based sampling, and randomized solver-based sampling for t=1, t=2 and t=3, with large effect sizes.

The pure distance-based sampling leads to higher error rates than diversified distance-based sampling and random sampling, for all sample sizes. However, comparing diversified distance-based sampling to random sampling, we see in the last row that random sampling has significantly lower error rates with small effect sizes for all sample sizes. The small effect sizes indicate that diversified distance-based sampling can reach nearly the same error rates as random sampling.

> For the property encoding time, uniform random sampling yields the most accurate performance models. Diversified distance-based sampling produces good results when a very limited number of samples are considered (*i.e.,* t=1) and almost reaches the accuracy of random when the sample sizes increase.

4.1.2 Input Sensitivity: Encoding Size. Random sampling and randomized solver-based sampling perform best to all other sampling strategies or similar to the best sampling for all input videos for sample sizes t=2 and t=3 (except for $x264_{14}$). For these sample sizes, most of the error rates of solver-based and diversified distance-based come close to random and randomized solver-based (*e.g.,* $x264_1$ and $x264_6$). For t=1, solver-based sampling leads to lower error rates for most of the inputs.

It is important to notice that the results in Table 4 are quite unstable compared to the results in Table 2. For example, random is the best sampling strategy for the input video $x264_{12}$, while a

F-test (p value)	Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
Solver-based															
Randomized solver-based	10^{-252}	10^{-125}							10^{-20}				10^{-13}		
Distance-based	10^{-238}	10^{-272}			10^{-38}								10^{-09}		
Diversified distance-based	10^{-259}				10^{-238}	10^{-89}	0.05	10^{-100}	10^{-179}				10^{-14}	10^{-04}	10^{-10}
Random	10^{-170}				10^{-195}	10^{-46}		10^{-71}	10^{-118}						

Table 6: p values from a one-sided pair-wise F-test for encoding time: we tested pair-wisely whether the variances of the error rate of the sampling from the row is significantly lower than the one from the column, for different sample sizes.

F-test (p value)	Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
Solver-based								10^{-94}							
Randomized solver-based	10^{-38}	10^{-69}	10^{-40}					10^{-290}	10^{-45}		10^{-13}				
Distance-based	10^{-75}			10^{-09}											
Diversified distance-based	10^{-89}	10^{-27}	10^{-136}	10^{-14}		10^{-34}	0.04	10^{-202}	10^{-144}						
Random	10^{-126}	10^{-75}	10^{-252}	10^{-31}		10^{-107}	10^{-09}	10^{-302}	10^{-263}	10^{-05}	10^{-15}	10^{-25}			

Table 7: p values from a one-sided pair-wise F-test for encoding size: we tested pair-wisely whether the variances of the error rate of the sampling from the row is significantly lower than the one from the column, for different sample sizes.

randomized solver-based sampling win for the input video $x264_1$, and t-wise sampling win for the input video $x264_{15}$. Also, the accuracy heavily depends on the sampling size. For example, for the input video $x264_{14}$, for t=1 randomized solver-based sampling yields the most accurate performance models; while for t=2 and t=3 solver-based outperforms the other sampling strategies.

In Table 5, we apply one-sided Mann-Whitney U tests pair-wisely to compare pairs of sampling strategies. The first row shows that t-wise sampling has a significantly lower error rate than distance-based sampling for t=1, t=2, and t=3 with small, large and medium effect sizes, respectively; t-wise sampling has also significantly lower error rates than diversified distance-based for t=1.

In the second row, we see that solver-based sampling leads to lower error rates than all other sampling strategies for t=1 with medium and large effect sizes. Solver-based, randomized solver-based, and diversified distance-based lead to lower error rates than t-wise for t=2 and t=3; and distance-based for t=1, t=2 and t=3. Overall, distance-based sampling leads to higher or similar error rates than other sampling strategies. Finally, we can notice that on average random leads to significantly lower error rates than all other strategies (except for solver-based sampling for t=1).

> For the property encoding size, random sampling and randomized solver-based sampling outperform all other sampling strategies for most of the input videos with sample sizes t=2 and t=3; and solver-based sampling outperforms for sample sizes t=1. Overall, random, randomized solver-based, solver-based, and diversified distance-based present good and similar accuracy for t=2 and t=3. Differently from our previous results (for time), there is not a clear winner.

4.2 Results RQ2—Robustness

We repeated each experiment 100 times for each sampling strategy and sample size, from which we obtained the distribution of mean error rates. We use the Levene's test to check whether there are significantly different variances between sampling strategies. In cases where we found significant variances, we performed pair-wisely one-sided F-tests. We show the results in Tables 6 and 7. Notice that we excluded t-wise sampling as it is a deterministic sampling and does not lead to any variations.

4.2.1 Encoding Time. In the second and third rows of Table 6, we can see that randomized solver-based sampling and distance-based sampling have a significantly lower variance than solver-based sampling for t=1 and t=2; the same also applies for random with t=1. In the two last rows, we observe that diversified distance-based and random have a lower variance than solver-based for t=1; both sampling have also lower variance than randomized solver-based and distance-based, for t=2 and t=3. Diversified distance-based has a significantly lower variance than random sampling for all sample sizes. Finally, random has a lower variance than t-wise for t=1; and randomized solver-based and distance-based for t=2 and t=3.

> For the property encoding time, diversified distance-based sampling is more robust than the other sampling strategies.

4.2.2 Encoding Size. In the first row of Table 7, we can see that solver-based sampling has a significantly lower variance than distance-based for t=2. In the second row, we can see that randomized solver-based sampling has a significantly lower variance than solver-based sampling on all sample sizes. Randomized solver-based sampling has also a significantly lower variance than distance-based sampling for t=2 and t=3; and diversified distance-based sampling for t=2. When it comes to the diversified distance-based sampling,

Video	Coverage-based			Solver-based			Randomized solver-based			Distance-based			Diversified distance-based			Random		
	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$	$t=1$	$t=2$	$t=3$
sintel_trailer ($x264_4$)	21.8 %	12.3 %	14.4 %	23.9 %	21.2 %	22.0 %	18.3 %	21.1 %	22.5 %	14.2 %	11.7 %	9.7 %	13.9 %	10.1 %	8.9 %	13.9 %	**9.4 %**	**8.8 %**
sintel_trailer (x264[25])	20.9 %	11.9 %	10.9 %	26.2 %	40.4 %	42.2 %	18.5 %	22.2 %	33.2 %	14.7 %	10.0 %	9.4 %	**12.6 %**	**8.8 %**	**9.0 %**	13.5 %	9.2 %	9.1 %

Table 8: Error rates of t-wise, (randomized) solver-based, (diversified) distance-based, and random sampling for the prediction of encoding time for the Sintel trailer input video (734 MB). The best results are highlighted in bold if the Mann-Whitney U test reported a significant difference ($p < 0.05$).

it leads to a significantly lower variance than the other sampling strategies, except for randomized solver-based (t=2) and random. In the last row, we observe that random has the lowest variance.

> For the property encoding size, uniform random sampling is more robust than the other sampling strategies.

5 DISCUSSION

The surprising effectiveness of coverage-based sampling. Why is t-wise sampling more effective for the input video $x264_{15}$? For $x264_{15}$, the performance distribution is rather unique compared to the others (see Figure 2b, page 4). To investigate why t-wise is more effective on this distribution, we analyzed the set of selected features by t-wise. We aim at understanding which features are frequently included in the sampling and how its frequencies differ from random. We recall that t-wise sampling is not subject to randomness and is the same for any video and for encoding time or size. We observed that the t-wise sampling prioritizes the feature `no_mbtree` to false, *i.e.,* `no_mbtree` is most of the time deactivated (it is not the case in random that has a good balance). To better understand the importance of `no_mbtree` for $x264_{15}$, we use (1) a random forest for computing so-called feature importance [7, 13, 37, 45] and (2) polynomial regressions for computing coefficients of options and pair-wise interactions among options [14, 50, 68]. We found that `no_mbtree` has a feature importance of 0.97 (out of 1) for encoding size (see our supplementary website). We also found that the feature `no_mbtree` has a strong negative effect on video $x264_{15}$. As for solver-based sampling, t-wise sampling relies on clustered zones (see Section 2.2) that luckily cover the most influential options. The clustering zones must contain features with high significance on performance which may explain high accuracy results. Specifically, the sample-set of t-wise is locally clustered around the configurations with `no_mbtree` to false, which is a very effective strategy for $x264_{15}$. However, for other input videos, the coverage criterion is not well-suited.

The surprising effectiveness of solver-based sampling. For the property encoding size, we observe good accuracy predictions for three sampling strategies: random, randomized solver-based, and solver-based. Why solver-based sampling strategies are sometimes more effective than random for size? We further analyzed the set of frequently selected features by these strategies for both experiments of time and size. We observe that the randomized solver-based and solver-based strategies were very effective to capture "clustered" effects of important options for size. In particular, randomized solver-based tend to follow a similar strategy as t-wise: `no_mbtree` option is most of the time false in the sample set. For t=1, `no_mbtree` option is even always set to false. The employed solver-based sampling strategy relies on a random seed which defines clustering zones (see Section 2.2) that luckily cover the most influential options. We did observe strong deviations of options' frequencies in the samples of solver-based sampling strategy compared to random (see our supplementary website). For the prediction of encoding time, the clustering mostly contain features with low significance which may explain the worst accuracy results.

There are several solvers available and each solver has its owns particularities, *i.e.,* the way they pick and thus cluster configurations can drastically change. The assumption that a certain sampling is the best one by comparing it to a solver-based sampling is not reliable since solver-based strategies may cluster by accident the (un-)important options to a specific performance property. Our study calls to propose sampling strategies with predictable and documented properties of the frequencies of options in the sample (*e.g.,* uniform random sampling or distance-based sampling).

Is there a "dominant" sampling strategy when the system performance is measured over the variation of hardware and the version of the target system? We compare our results with the results obtained in Kaltenecker et al. [25] to encode the *same* ($x264_4$) input video Sintel trailer (734 MB). Their measurements were performed on different hardware (Intel Core Q6600 with 4 GB RAM (Ubuntu 14.04)) and over a different version of x264. Diversity distance-based sampling yields the best performance in their settings. In our case, random sampling outperforms diversity distance-based sampling (see Table 8). Thus, we can hypothesize that *sampling strategies are also sensitive to different hardware characteristics and software versions (in addition to targeted performance properties and input videos).* Further experiments on different hardware and versions are needed though to confirm this hypothesis. There are several questions that arise from this: To what extent are sampling strategies sensitive to different hardware characteristics and software versions? Are the interactions and influence of configuration options on performance consistent across different software versions? Do we experience the same for different performance properties?

Answering RQ1 (accuracy) Is there a "dominant" sampling strategy for the x264 configurable system whatever inputs and targeted quantitative properties? For the property encoding time, there is a dominant sampling strategy (*i.e.,* uniform random sampling) as shown in Kaltenecker et al. [25], and thus the sampling can be reused whatever the input video is. For the property encoding size, although the results are similar around some sampling strategies, they differ in a noticeable way from encoding time and suggest a higher input sensitivity. Overall, random is the state-of-the-art sampling strategy but is not a dominant sampling strategy in all

cases, *i.e.,* the ranking of dominance changes significantly given different inputs, properties and sample sizes. A possible hypothesis is that individual options and their interactions can be more or less influential depending on input videos, thus explaining the variations' effect of sampling over the accuracy. Our results pose a new challenge for researchers: Identifying what sampling strategy is the best given the possible factors influencing the configurations' performances of a system.

Answering RQ2 (robustness). We have quantitatively analyzed the effect of a sampling strategy over the prediction variance. Overall, random (for size) and diversified distance-based (for time and size) have higher robustness. We make the observation that uniform random sampling is not necessarily the best choice when robustness should be considered (but it is for accuracy). In practical terms, practitioners may have to find the right balance between the two objectives. As a sweet-spot between accuracy and robustness, diversified distance-based sampling (for time), and either random or randomized solver-based sampling (for size) are the best candidates. We miss however an actionable metric that could take both accuracy and robustness into account.

Our recommendations for practitioners are as follows:

- uniform random sampling is a very strong baseline whatever the inputs and performance properties. In the absence of specific-knowledge, practitioners should rely on this dominant strategy for reaching high accuracy;
- in case uniform random sampling is computationally infeasible, distance-based sampling strategies are interesting alternatives;
- the use of other sampling strategies does not pay off in terms of prediction accuracy. When robustness is considered as important, uniform random sampling is not the best choice and here we recommend diversified distance-based sampling.

The impacts of our results on configuration and performance engineering research are as follows:

- as uniform random sampling is effective for learning performance prediction models, additional research effort is worth doing to make it scalable for large instances. Recent results [19, 38, 44, 47] show some improvements, but the question is still open for very large systems (*e.g.,* Linux [1]);
- some sampling strategies are surprisingly effective for specific inputs and performance properties. Our insights suggest the existence of specific sampling strategies that could prioritize specific important (interactions between) options. An open issue is to discover them for any input or performance property;
- performance measurements with similar distributions may be grouped together to enable the search for dominants sampling strategies;
- beating random is possible but highly challenging in all situations;
- it is unclear how factors such as the version or the hardware influence the sensitivity of the sampling effectiveness (and how such influence differs from inputs and performance properties);
- we warn researchers that the effectiveness of sampling strategies for a given configurable system can be biased by the workload and the performance property used.

6 THREATS TO VALIDITY

Despite the effort spent on the replication of the experiments in Kaltenecker et al. [25], we describe below some *internal* and *external* threats to the validity of this study.

Internal Validity. A threat to the internal validity of this study is the selection of the learning algorithm and its parameter settings which may affect the performance of the sampling strategy. We used the state-of-the-art step-wise multiple linear regression algorithm from [52] which has shown promising results in this field [25, 46]. However, we acknowledge that the use of another algorithm may lead to different results. Still, conducting experiments with other algorithms and tuning parameters is an important next step, which is part of our future work.

As another internal threat to validity, our results can be subject to measurement bias. While the property encoding size is stable, the measurement of the property encoding time deserves careful attention. To mitigate any measurement bias, we measured the time from all different input videos multiple times and used the average in our learning procedure. Also, we control external factors like the hardware and workload of the machine by using a grid computing infrastructure called IGRIDA[2]. We take care of using the same hardware and mitigate network-related factors. Using a public grid instead of a private cloud allows us to have control of the resources we used for measuring time. To additionally quantify the influence of measurement bias on our results, we also analyzed their Relative Standard Deviation to prove we have a stable set of measurements.

To assess the accuracy of the sampling strategies and the effect of different inputs, we used MRE since most of the state-of-the-art works use this metric [46]. As in [25], we compute MRE based on the whole population of all valid configurations, *i.e.,* we include the sample used as training to the testing set for comparing sampling strategies. However, according to several authors [16, 40, 41, 46][51, 58–60] the prediction error rate on new data is the most important measurement, *i.e.,* the error rate should not be computed over the training set t used to build the model. As future work, we plan to explore the effectiveness between four well-established resampling methods [46]: hold-out, cross-validation, bootstrapping, and dynamic sector validation.

External Validity. A threat to external validity is related to the used case study and the discussion of the results. Because we rely on a specific system and two performance properties, the results may be subject to this system and properties. However, we focused on a single system to be able at making robust and reliable statements about whether a specific sampling strategy can be used for different inputs and performance measurements. We conducted a discussion with all authors of the paper to avoid any confusing interpretation or misunderstanding of the results. Once we are able to demonstrate evidences to x264, we can then perform such an analysis also over other systems and properties to generalize our findings.

[2] http://igrida.gforge.inria.fr/

We set our dataset to up seventeen input videos and two properties due to budget restrictions. As with all studies, there is an inherent risk to not generalize our findings for other input videos and performance properties (*e.g.,* energy consumption). While the seventeen videos cover a wide range of different input particularities and provided consistent results across the experiments, this is a preliminary study in this direction and future work should consider additional inputs based on an in-depth qualitative study of video characteristics.

7 CONCLUSION

Finding a small sample that represents the important characteristics of the set of all valid configurations of a configurable system is a challenging task. Numerous sampling strategies have been proposed [46] and a recent study [25] have designed an experiment to compare six sampling strategies. However, this study did not investigate whether the strategies developed so far generalize across different inputs and performance properties of the same configurable system. To this end, we replicated this study to investigate the individual effects of 17 input videos on the prediction accuracy of two performance properties. Our results demonstrated that uniform random sampling dominates for most input videos and both performance properties, time and size. There are some cases for which random sampling can be beaten with specific sampling. However, for the other sampling strategies, the prediction accuracy (*i.e.,* the ranking of sampling strategies) can dramatically change based on the input video and sampling size. It has practical implications since users of configurable systems feed different inputs and deal with different definitions of performance. Thus, we warn researchers about the sensitivity of a sampling strategy over workloads and performance properties of a configurable system. This work provides a new view of random sampling based on its effective dominance across different inputs and performance properties. Distance-based sampling strategies are other relevant alternatives. Our replication of the original experiment design is a promising starting point for future studies of other configurable systems, consideration of other influential factors (versions, hardware), and the use of specific (*e.g.,* white-box) sampling strategies.

ACKNOWLEDGMENTS

This research was funded by the ANR-17-CE25-0010-01 VaryVary project. We thank the anonymous reviewers of ICPE. We also thank Arnaud Blouin, Luc Lesoil, and Gilles Perrouin for their comments on an early draft of this paper.

REFERENCES

[1] Mathieu Acher, Hugo Martin, Juliana Alves Pereira, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Eddine Khelladi, Luc Lesoil, and Olivier Barais. 2019. Learning Very Large Configuration Spaces: What Matters for Linux Kernel Sizes. https://hal.inria.fr/hal-02314830 Research Report, Inria.

[2] Mathieu Acher, Paul Temple, Jean-Marc Jezequel, José A Galindo, Jabier Martinez, and Tewfik Ziadi. 2018. VaryLaTeX: Learning Paper Variants That Meet Constraints. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, 83–88.

[3] Benoit Amand, Maxime Cordy, Patrick Heymans, Mathieu Acher, Paul Temple, and Jean-Marc Jézéquel. 2019. Towards Learning-Aided Configuration in 3D Printing: Feasibility Study and Application to Defect Prediction. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, 7.

[4] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, USA, 1–10.

[5] Andrea Arcuri and Lionel Briand. 2012. Formal Analysis of the Probability of Interaction Fault Detection Using Random Testing. *IEEE Transactions on Software Engineering* 38, 5 (Sept 2012), 1088–1099.

[6] Liang Bao, Xin Liu, Ziheng Xu, and Baoyin Fang. 2018. AutoConfig: Automatic Configuration Tuning for Distributed Message Systems. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 29–40.

[7] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[8] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In *Tools and Algorithms for the Construction and Analysis of Systems TACAS'15 2015, London, UK, April 11-18, 2015. Proceedings*. 304–319.

[9] Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. 2013. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *International Conference on Computer Aided Verification*. Springer, 608–623.

[10] Shiping Chen, Yan Liu, Ian Gorton, and Anna Liu. 2005. Performance Prediction of Component-Based Applications. *Journal of Systems and Software* 74, 1 (2005), 35–43.

[11] Myra B Cohen, Matthew B Dwyer, and Shi, Jiangfan. 2008. Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. *IEEE TSE* 34, 5 (2008), 633–650.

[12] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[13] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2018. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. arXiv:arXiv:1801.01489

[14] Alexander Grebhahn, Carmen Rodrigo, Norbert Siegmund, Francisco J Gaspar, and Sven Apel. 2017. Performance-Influence Models of Multigrid Methods: A Case Study on Triangular Grids. *Concurrency and Computation: Practice and Experience* 29, 17 (2017), e4057.

[15] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-Aware Performance Prediction: A Statistical Learning Approach. In *Automated Software Engineering (ASE)*. IEEE, 301–311.

[16] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu. 2017. SMTIBEA: A Hybrid Multi-Objective Optimization Algorithm for Configuring Large Constrained Software Product Lines. In *Software & Systems Modeling*.

[17] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2018. Test them All, Is It Worth It? Assessing Configuration Sampling on the JHipster Web Development Stack. *Empirical Software Engineering*.

[18] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. 2014. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines. *IEEE Trans. Software Eng.* (2014).

[19] Ruben Heradio, David Fernández-Amorós, Christoph Mayr-Dorn, and Alexander Egyed. 2019. Supporting the Statistical Analysis of Variability Models. In *41st International Conference on Software Engineering, ICSE*. 843–853.

[20] Pooyan Jamshidi, Javier Cámara, Bradley Schmerl, Christian Kästner, and David Garlan. 2019. Machine Learning Meets Quantitative Planning: Enabling Self-Adaptation in Autonomous Robots. *arXiv preprint arXiv:1903.03920* (2019).

[21] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 497–508.

[22] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to Sample: Exploiting Similarities Across Environments to Learn Performance Models for Configurable Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 71–82.

[23] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 31–41.

[24] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2012. An Algorithm for Generating t-Wise Covering Arrays from Large Feature Models. In *Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -volume 1*, Vol. 1. ACM, 46.

[25] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-Based Sampling of Software Configuration Spaces. In *Proceedings of the International Conference on Software Engineering (ICSE)*.

[26] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, and Sven Apel. 2017. On the Relation of External and Internal Feature Interactions: A Case Study. *arXiv preprint arXiv:1712.07440* (2017).

[27] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. 2019. Tradeoffs in Modeling Performance of Highly Configurable Software Systems. *Software & Systems Modeling* 18, 3 (01 Jun 2019), 2265–2283. https://doi.org/10.1007/s10270-018-0662-9

[28] Thomas Krismayer, Rick Rabiser, and Paul Grünbacher. 2017. Mining Constraints for Event-Based Monitoring in Systems of Systems. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 826–831.

[29] William H Kruskal and W Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *Journal of the American statistical Association* 47, 260 (1952), 583–621.

[30] D.R. Kuhn, D.R. Wallace, and A.M. Gallo. 2004. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering* 30, 6 (jun 2004), 418–421.

[31] Daniel Le Berre and Anne Parrain. 2010. The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation* 7 (2010), 59–64. https://hal.archives-ouvertes.fr/hal-00868136

[32] Howard Levene. 1961. Robust Tests for Equality of Variances. *Contributions to probability and statistics. Essays in honor of Harold Hotelling* (1961), 279–292.

[33] Max Lillack, Johannes Müller, and Ulrich W Eisenecker. 2013. Improved Prediction of Non-Functional Properties in Software Product Lines with Domain Context. *Software Engineering 2013* (2013).

[34] Henry B Mann and Donald R Whitney. 1947. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *The annals of mathematical statistics* (1947), 50–60.

[35] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. 2016. A Comparison of 10 Sampling Algorithms for Configurable Systems. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. ACM Press, Austin, Texas, USA, 643–654.

[36] Marcilio Mendonca, Andrzej Wasowski, Krzysztof Czarnecki, and Donald Cowan. 2008. Efficient Compilation Techniques for Large Scale Feature Models. In *Int'l Conference on Generative programming and component engineering*. 13–22.

[37] Christoph Molnar. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.

[38] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don S. Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models that Have Numerical Features. In *International Systems and Software Product Line Conference (SPLC)*. 39:1–39:13.

[39] I Made Murwantara, Behzad Bordbar, and Leandro L. Minku. 2014. Measuring Energy Consumption for Web Service Product Configuration. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services (iiWAS)*. ACM, New York, NY, USA, 224–228.

[40] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using Bad Learners to Find Good Configurations. In *Proceedings of the European Software Engineering Conference/Foundations of Software Engineering (ESEC/FSE)*. 257–267.

[41] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Faster Discovery of Faster System Configurations with Spectral Learning. *Automated Software Engineering* (2018), 1–31.

[42] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding Faster Configurations Using Flash. *IEEE Transact. on Software Engineering* (2018).

[43] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-Optimal Configurations in Product Lines by Random Sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. 61–71.

[44] Jeho Oh, Paul Gazzillo, and Don S. Batory. 2019. *t*-Wise Coverage by Uniform Sampling. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France*. 15:1–15:4.

[45] Terence Parr, Kerem Turgutlu, Christopher Csiszar, and Jeremy Howard. 2018. Beware Default Random Forest Importances. last access: july 2019.

[46] Juliana Alves Pereira, Hugo Martin, Mathieu Acher, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2019. Learning Software Configuration Spaces: A Systematic Literature Review. arXiv:arXiv:1906.03018

[47] Quentin Plazar, Mathieu Acher, Gilles Perrouin, Xavier Devroey, and Maxime Cordy. 2019. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet?. In *International Conference on Software Testing, Verification, and Validation (ICST)*. 1–12.

[48] Adam Porter, Cemal Yilmaz, Atif M Memon, Douglas C Schmidt, and Bala Natarajan. 2007. Skoll: A Process and Infrastructure for Distributed Continuous Quality Assurance. *IEEE Transactions on Software Engineering* 33, 8 (2007), 510–525.

[49] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2016. Towards Predicting Feature Defects in Software Product Lines. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*. ACM, 58–62.

[50] Faiza Samreen, Yehia Elkhatib, Matthew Rowe, and Gordon S Blair. 2016. Daleel: Simplifying Cloud Instance Selection Using Machine Learning. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 557–563.

[51] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems (T). In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 342–352.

[52] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kastner. 2015. Performance-Influence Models for Highly Configurable Systems. In *10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 284–294.

[53] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don S. Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting Performance via Automated Feature-Interaction Detection. In *International Conference on Software Engineering (ICSE)*. 167–177.

[54] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, and Gunter Saake. 2008. Measuring Non-Functional Properties in Software Product Line for Product Derivation. In *2008 15th Asia-Pacific Software Engineering Conference*. IEEE, 187–194.

[55] George W Snedecor and Witiiam G Cochran. 1989. Statistical Methods, 8thEdn. *Ames: Iowa State Univ. Press Iowa* (1989).

[56] Charles Song, Adam Porter, and Jeffrey S Foster. 2013. iTree: Efficiently Discovering High-Coverage Configurations Using Interaction Trees. *IEEE Transactions on Software Engineering* 40, 3 (2013), 251–265.

[57] Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.* 27, 3, Article 11 (Sept. 2018), 51 pages.

[58] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, and Olivier Barais. 2017. Learning Contextual-Variability Models. *IEEE Software* 34, 6 (2017), 64–70.

[59] Paul Temple, Mathieu Acher, Gilles Perrouin, Battista Biggio, Jean-Marc Jézéquel, and Fabio Roli. 2019. Towards quality assurance of software product lines with adversarial configurations. In *23rd International Systems and Software Product Line Conference, SPLC*. ACM, 38:1–38:12.

[60] Paul Temple, José Angel Galindo Duarte, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using Machine Learning to Infer Constraints for Product Lines. In *Software Product Line Conference (SPLC)*. Beijing, China. https://hal.inria.fr/hal-01323446

[61] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 847–855.

[62] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *Comput. Surveys* (2014).

[63] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. 2015. Empirical Comparison of Regression Methods for Variability-Aware Performance Prediction. In *SPLC'15*.

[64] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 39–50.

[65] András Vargha and Harold D Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[66] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A Classification of Product Sampling for Software Product Lines. In *International Systems and Software Product Line Conference (SPLC)*. 1–13.

[67] Markus Weckesser, Roland Kluge, Martin Pfannemüller, Michael Matthé, Andy Schürr, and Christian Becker. 2018. Optimal Reconfiguration of Dynamic Software Product Lines Based on Performance-Influence Models. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 98–109.

[68] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. 2012. Automated Inference of Goal-Oriented Performance Prediction Functions. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 190–199.

[69] Cemal Yilmaz, Myra B Cohen, and Adam A Porter. 2006. Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces. *IEEE Transactions on Software Engineering* 32, 1 (2006), 20–34.

[70] Yi Zhang, Jianmei Guo, Eric Blais, Krzysztof Czarnecki, and Huiqun Yu. 2016. A Mathematical Model of Performance-Relevant Feature Interactions. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 25–34.

[71] Wei Zheng, Ricardo Bianchini, and Thu D Nguyen. 2007. Automatic Configuration of Internet Services. *ACM SIGOPS Operating Systems Review* 41, 3 (2007), 219–229.

Sampling Effect on Performance Prediction of Configurable Systems: A Case Study (Artifact)

ABSTRACT

This document contains a stable URL to the artifacts associated with the paper "Sampling Effect on Performance Prediction of Configurable Systems: A Case Study" and provides general instructions to reproduce the presented results and reuse the datasets.

1 STABLE URL WITH THE ARTIFACTS

All resources, artifacts, and detailed procedures to replicate the paper results are available at github.com/jualvespereira/ICPE2020[1].

2 RESOURCES AND ARTIFACTS

The README.md file contains a detailed roadmap on the technical requirements of how to set up and run the experiments. It is structured into two parts: *(1)* performance prediction, and *(2)* aggregation and visualization of results.

2.1 Performance Prediction

To reproduce our results, users must follow the steps below:

- Install Docker: https://docs.docker.com/install/
- Download the container: `docker pull hmartinirisa/icpe2020:latest` (by invoking this script, all required ressources are installed, which might take several minutes).
- Run the container: `sudo docker run -it -v "$(pwd)":/docker hmartinirisa/icpe2020 bash`
- Go either to the directory `Distance-Based_Data_Time` or `Distance-Based_Data_Size`: `cd ICPE2020/Distance-Based_Data_Time` or `cd ICPE2020/Distance-Based_Data_Time`.
- For each `<sampling-approach>` (twise, solvBased, henard, distBased, divDistBased, rand) and `<case-study>` ($x264_0$, $x264_1$,..., $x264_{16}$), run the script: `./SPLConquerorExecuter.py <case-study> <sampling-approach> <save-location>` (example: `./SPLConquerorExecuter.py x264_0 rand /docker/ICPE2020/DistanceBased_Data_Time/SupplementaryWebsite/PerformancePredictions/AllExperiments`). The experiments will run for 100 random seeds.

Dependencies. The implementation depends on the SPL Conqueror for sampling and learning, and on the z3 Constraint solver to navigate through the configuration space of the configurable system.

Main repository. The main resources and artifacts of this paper are divided into two directories: `Distance-Based_Data_Time` and `Distance-Based_Data_Size`. Each directory contains the data related to *time* and *size* prediction, respectively. `SPLConquerorExecuter.py` is the main script to run the experiments and compute the prediction error rates. As input to this script, we need a feature model and the measurements of performance for the input video being considered. The directory `SupplementaryWebsite/MeasuredPerformanceValues` contains the data for all seventeen analysed input videos, namely $x264_0$, $x264_1$, $x264_2$, ..., $x264_{16}$.

- *FeatureModel.xml*: file containing the description of the configurable system $x264$.
- *measurements.xml*: file containing the measured performance values for all valid configurations of an specific input video.
- *input_video.txt*: file containing the description of the input video we used to measure all valid configurations.

To perform experiments related to a new case study *case_x*, the user must add the case-study name *case_x* into the files `SPLConquerorExecuter.py` and `analyzeRuns.py`, and add the files `FeatureModel.xml` and `measurements.xml` at a new folder *case_x* into the directory `Distance-Based_Data_Time/SupplementaryWebsite/MeasuredPerformanceValues` (as an example see these files for $x264$ case study – more information about the format of these files can be also found at github.com/se-passau/SPLConqueror).

The directory `SupplementaryWebsite/PerformancePredictions/AllExperiments` contains a set of prediction log files with the experiment results (*i.e.*, error rate). Each sampling error rate is computed 100 times using different random seeds for each input video.

- *learn_<sampling-approach>_<sampling-size>.a*: file containing a list of all input commands used to run SPL Conqueror[2].
- *out_<sampling-approach>_<sampling-size>.log*: file containing the error rate of applying a sampling strategy with a certain sample size on an input video (the error rate is the last number in the last line before *"Analyze finished"*).
- *sampledConfigurations_<sampling-approach>_<sampling-size>.csv*: file containing the set of configurations used as input sample to the machine-learning technique.

2.2 Aggregation and Visualization of Results

`analyzeRuns.py` and `ErrorRateTableCreator.py` are the main scripts to aggregate and visualize the results. `analyzeRuns.py` collects all error rates from all 100 runs of all case studies in an unique file *all_error_<sampling-approach>_<sampling-size>.txt* (see directory `PerformancePredictions/AllSummary`). Then, the script `ErrorRateTableCreator.py` reads this file and uses `PerformKruskalWallis.R` to generate the visualization data available in Tables 2-8 from the paper (see the output tex-files at the directory `latex`).

- `./analyzeRuns.py <run-directory> <output-directory>`
- `./ErrorRateTableCreator.py <input-directory> <sampling-approaches> <labels> <output-tex>`

`<run-directory>` is the directory where all runs of all case studies are stored. `<output-directory>` and `<input-directory>` are the directories where the aggregated results should be written to and read from, respectively[3]. `<sampling-approaches>` and `<labels>` contain the list of sampling approaches to consider and the labels that should be used in the table. `<output-tex>` contains the directory where the tex-files should be written to[4].

[1] These are proprietary artifacts (except the datasets with all computed measurements) from github.com/se-passau/Distance-Based_Data with slightly modifications.

[2] For detailed explanation on these commands see github.com/se-passau/SPLConqueror

[3] In the replication process, the generated error rates on this directory must be the same as the provided ones on the directory `AllSummary` of our GitHub repository for the same sampling approach, sample sizes and random seeds.

[4] see these command lines example at our repository (README.md).