Learning Once and for All? On the Input Sensitivity of Configurable Systems

Anonymous submission

***Abstract*—Widely used software systems such as compilers or 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.,* generate code as compact as possible or compress a video down to a certain size. Several research work have thus proposed to resort on machine learning to predict the effect of options on such functional and performance properties. However these approaches fail to account for the variability of inputs to these software systems (e.g., a program fed to a compiler like gcc or a video as input to an encoder like x264), making their predictions close to worthless for a field usage. In this paper, we first conduct a large study over 1300+ inputs fed to 200+ software configurations of *x264* that shows the significance of the input sensitivity problem on 5 performance properties. To overcome this input sensitivity, we propose *Inputec*, a novel technique for learning a near-optimal configuration from a representative set of inputs. The techniques produces as output an offline engine capable of generating a suitable configuration for the task at hand. The novelty of this technique lies in the fact that *Inputec* only uses input characteristics to choose a configuration, without additional learning. We show how *Inputec* can efficiently help engineers in finding configurations for *x264* that most often outperform those obtained from state of the art techniques by about 25%.**

I. Introduction

Widely used software systems such as compilers or video encoders are by necessity highly configurable, with hundreds or even thousands of options to choose from. For example, a compiler like *gcc* offers different options such as *-O1* or *-Wall* for compiling a program. 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 these options (*i.e.,* finding a proper *configuration* of the software system) to meet their goals for the tasks at hand, *e.g.,* generate code as compact as possible 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 having a high-dimensional variability space [1].

Research work have shown that predicting the performance of configurations with accuracy using a reasonable sample of measurements is possible [2]–[6]. These works measure performance of a configurations sample under specific settings to then build a model capable of predicting the performance of all configurations, *i.e.,* a performance model. However, inputs (*e.g.,* a program fed to a compiler like *gcc* or a video provided as input to an encoder like *x264*) can also strongly impact the performances of a configurable system [7]. The *x264* encoder typifies this problem. For example, Kate, an engineer working for a VOD company, wants an *x264* performance model for predicting the size of output videos based on selected *x264* configuration options. She would build such a model using several input videos to learn about the impact of these *x264* configuration options on the output video size. Now, if Kate wants to reuse this model for new input videos, she might ask herself the following questions. Will this performance model predict an accurate value for the encoded size? Do configuration options have the same effect on the encoded size despite a different input? Do options interact in the same way no matter the inputs? These are crucial practical issues since inputs fed to configurable systems can question performance prediction models developed so far. In particular, prior research works ignore whether these prediction models generalize across different inputs and thus can be reused. If they cannot, Kate would have to train a new performance model, the worst situation being to learn as many performance models as there are inputs.

In this work, we first conduct an in-depth empirical study on the impact of inputs on a large configurable system: the *x264* video encoder. We chose *x264*, the most established open-source software encoder for *H.264/AVC*, since input sensitivity is a well-known problem in the video compression community [8]. Numerous related research work has been done on *x264* and may benefit from our results [2]–[6], [9]–[11]. But to the best of our knowledge, none of these work provides a comprehensive study of the impact of the input fed to *x264*. We systematically explore the impacts of 1397 input videos from the YouTube User General Content (UGC) dataset on five quantitative properties of 201 configurations of *x264*. It reveals that input videos can interact with software options, significantly impacting the performances of *x264*.

To overcome this input sensitivity, we propose *Inputec*, a novel technique for learning a near-optimal configuration from a representative set of inputs. The technique produces as output an offline engine capable of generating a good configuration for engineers (so they do not have to do it manually). The novelty of this technique lies in the fact that *Inputec* only uses input properties to predict a configuration, without additional learning. We show how *Inputec* can efficiently help engineers in finding configurations for *x264* that most often outperform those obtained from state of the art techniques by about 25%.

In summary, our contributions are as follows:

- We conducted, to our best knowledge, the first large empirical study that investigates the impact of input on performances of a configurable system, namely *x264*;
- We develop an approach, *Inputec*, for predicting a configuration minimizing or maximizing a software performance, adapted for an input video, and evaluate it on 1397 inputs;
- We extend Maxiaguine *et al.* video classification [12] and draw out four encoding profiles of videos influencing the *bitrate* of video compression;
- For replicability, we provide a comprehensive dataset[1] of configurations' measurements as well as learning procedures[2].

The remainder of this paper is organized as follows: Section II explores the problem of input sensitivity, Section III explains our approach, Section IV evaluates it, Section V discusses its limitations, Section VI details the threats to validity, Section VII presents related work and Section VIII concludes our paper.

II. Sensitivity to Inputs of *x264*: an Empirical Study

This section details an in-depth study of *x264*, a command line utility allowing to encode or transcode video files or streams; users can configure *x264* by selecting configuration options, adding or removing functionalities. Numerous organizations (*e.g., Netflix* [13]) rely on this widely-used video compression standard, with ubiquitous decoder support on Web browsers, TVs, or mobile devices. Moreover, prior research work considered the *x264* configurable system [2]–[6], [10] for predicting the execution time, using a unique or a few videos for performing measurements. A possible pitfall is that such predictions may not be reusable for a large panel of input videos. To quantify this pitfall the study synthesizes evidences to answer the following main research question:

RQ1. Do Input Videos Change Performances of x264 Configurations?

A. Research Questions

We divide RQ1 into the two following research questions.

RQ1.1 - Do software performances stay consistent across inputs?

A common assertion in the transfer learning community [14]–[16] is that the more the source and the target environment are similar in terms of software performances, the more the transfer will outperform the simple machine learning (*i.e.,* directly on the target environment, without transfer). However, inputs can have different properties that may change the software behavior and thus alter the software performance across inputs.

To check this hypothesis, we first compute, analyze and compare the Spearman correlation [17] of each couple of videos; the correlations are considered as a measure of similarity between the configurations' performances over two videos. We use the Evans rule [18] to interpret these correlations. In absolute value, we refer to correlations by the following labels; very weak: 0-0.19, weak: 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 weak or negative scores have practical implications: good configurations for a video can well exhibit bad performances for another video.

Second, we investigate the performances of the two configurations the most and least sensitive to input videos. Since it is not possible to directly compare two input performance distributions, for all the videos, we sort the n configurations of our dataset by increasing performances and assign a rank to each configuration (*i.e.,* 1 being the smallest and n the biggest values). We present the results of two configurations that have the maximal (*i.e.,* most sensitive to input videos) and minimal (*i.e.,* least sensitive to input videos) performance ranking standard deviations.

RQ1.2 - Are there some configuration options more sensitive to input videos?

Configuration options have different influences on *x264* performances (*e.g., time* or *size*). An option is called *influential* for a performance when its values have a strong effect on this performance [16], [19]. For example, developers may wonder whether the feature they add to a configurable software has an influence on its performance. Even if they know how to estimate qualitatively which features are influential, experts cannot always quantify it precisely, especially in high-dimensional feature spaces. Users also have to identify and leverage the influential options to configure *x264* to obtain a specific performance. However, is an option identified as influential for some videos, still influential for other videos? If not, it becomes both tedious and time-consuming to find the influential options for each video. To assess the relative importance of configuration options, we use two different methods to predict the performance values:

Random Forest. The tree structure provides insights about the most essential options for prediction as a tree splits first options that provide the highest information gain. We use random forests (a vote between multiple decision trees): we can derive, from the forests trained on the inputs, estimates of the options importances. For a random forest, we consider that a feature is influential if its feature importance (*i.e.,* median on all inputs) is greater than $\frac{1}{n_F}$, where n_F is the number of features considered in the dataset. This threshold represents the theoretic importance of features for a software having equally important features (analog to the Kaiser rule [20] with Principal Component Analysis).

Linear Regression. The coefficients of an Ordinary Least Square (OLS) regression weight the importance of configuration options. These coefficients can be positive (resp. negative) if a bigger (resp. lower) feature value results in a bigger performance value for the output video. Ideally, the sign of the coefficients of a same configuration should remain the same for all videos.

[1]The dataset is available on zenodo : https://zenodo.org/record/3928253

[2]Code and resources are available on github : https://anonymous.4open.science/r/df319578-8767-47b0-919d-a8e57eb67d25/

B. Protocol

Fixed hardware and operating system. We used a unique private server with the following configuration: 88 CPUs, model : Intel(R) Xeon(R) Gold 6238 CPU @ 2.10 GHz running over Linux Ubuntu 18.04.

Input. The YouTube **U**ser **G**eneral **C**ontent (UGC) dataset [21] of 1397 videos (20 seconds clips), dedicated to the study of video compression. We selected this dataset as it provides additional data about the content of videos; categories of video (such as Animation, Gaming, Vertical, Vlog, *etc.*) and other properties (*e.g.,* resolution, temporal complexity, spatial complexity, *etc.*). Besides, and according to Wang *et al.*, this dataset, extracted from a larger subset of millions of videos, highly represents the content uploaded to Youtube.

Software. *x264*, version 0.152.2854, git commit e9a5903. *x264* provides 10 preset configurations (see *x264* documentation), designed to work efficiently to optimize the encoding time of the process (ultrafast, fast, *etc.*) or the quality (slow, placebo, *etc.*) of the input video. We use a set of **p = 24** configuration options changed by the *preset* configuration options. For the 10 *presets*, we create 20 new configurations by modifying the *preset* configuration options (*i.e.,* changing one or two values of configuration options). We limited ourselves to 201 configurations due to budget constraints.

Performance measurement. The performance measurement protocol, and the different steps to reproduce the experience are detailed in the repository. For 201 configurations and 1397 videos, we configured *x264* and compressed the video. We systematically measured:

- *encoding time*: the elapsed time of the compression, how many seconds does it take for *x264* to encode a video, *i.e.,* to convert a given input video into an output video in the H.264 compressed format;
- *encoding size*: the size (in bytes) of the compressed video. Users typically expect to decrease the size of an input video while keeping a decent quality.

Additionally, *x264* provides metrics computed during the encoding process that we integrate to the dataset:

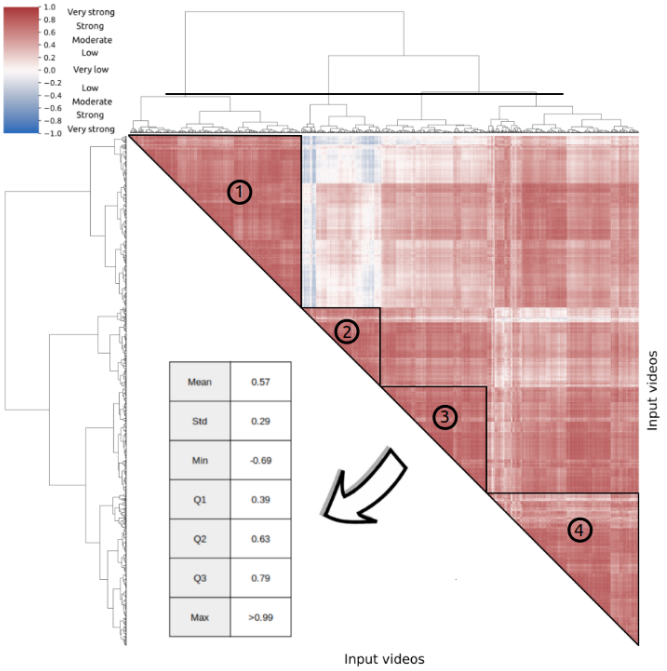
- *cpu* : the percentage of **c**entral **p**rocessing **u**nit used during the compression (can exceed 100 % with multi-core systems);
- *bitrate* : the average amount of data encoded per second;
- *fps* : the average number of **f**rames encoded **p**er **s**econd.

For the remainder of this paper, we adopt the following formatting; *configuration option* will be relative to a software configuration option (*i.e.,* a feature), input property to input properties (*i.e.,* constant for a given video), and *performance* to non functional properties of the software (*i.e.,* vary with the different configuration options).

C. Results

RQ1.1 - Do software performances stay consistent across inputs?

We present the results for the 5 performance properties w.r.t. the methodology described in Section II-A.

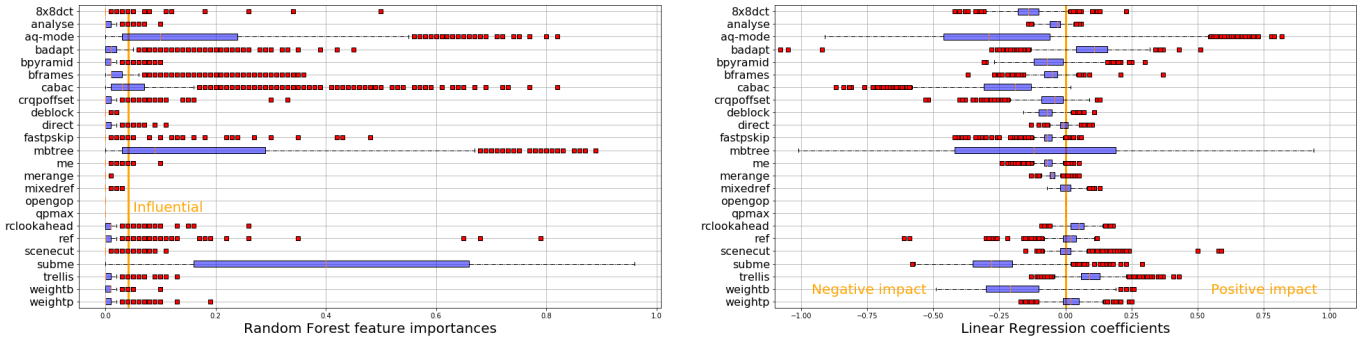


Each square$_{(i,j)}$ represents the Spearman correlation between the *bitrates* of the videos i and j. The color of this square respects the top-left scale: high positive correlations are red; low in white; negative in blue. Because we cannot describe each correlation individually, we added a table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

Fig. 1: Spearman correlogram of bitrates.

- for *encoding time*, we observe very strong correlations (0.93 on average and for first quartile). Though there are some exceptions among pairs of videos with very weak correlations (0.15), the input sensitivity is low for this property;
- for *fps*, we observe again very strong correlations (0.93 on average and for first quartile). Even extreme cases exhibit moderate correlations (0.50): we can conclude that the input sensitivity is low;
- for *cpu*, correlations are moderate (0.79 on average, first quartile 0.74) while there are negative correlations. Hence the input sensitivity is more important than for *encoding time* or *fps*;
- for *bitrate*, the input sensitivity is high with moderate correlation on average (0.57), weak correlation for first quartile (0.39), and some negative correlations (up to -0.69);
- for *encoding size* the input sensitivity is high and similar as *bitrate*.

Overall, depending on the considered software performance, the input sensitivity can be high (*e.g., bitrate* and *encoding size*) or low (*e.g., encoding time*, *encoding fps* and *cpu consumption*). Due to page limits, we now focus on the results for *bitrate*. The input sensitivity is indeed the most pronounced amongst the performance properties. In such a case, the performances of configurations unlikely generalize well among the videos.



(a) Feature importances

(b) Features impacts

Fig. 2: Feature importances and impacts - Bitrate

In Figure 1, we perform hierarchical clustering [22] to gather inputs having similar *bitrates* distributions and visually group together correlated-videos. Results suggest a positive correlation (see dark red cells), though there are pairs of videos with lower (see white cells) and even negative correlations (see dark blue cells). We can notice that:

- On the positive side, half of the videos have strong positive correlations between their *bitrates* distributions (*i.e.,* 0.63), and a quarter of the videos have very strong positive correlations (Q3 = 0.79).
- There are, however, less favorable cases with lower and negative coefficients: the worst case being -0.69, between video 9 and video 503; those negative results suggest that some videos have very different performance distributions.

We now focus on the most and least sensitive configurations. The rankings of *bitrates* for the configuration 200 are stable, whatever the video; 92.5 % of them are between 105 and 130 (limits excluded). Oppositely, configuration 16 rankings vary across inputs. There is no global trend to follow when ranking this configuration, sparse on the space of rankings (Q1: 54, Q2 : 103, Q3 : 171). It can be both one of the best configurations for minimizing the *bitrate* (*e.g.,* videos 147 or 551) and one of the worst (*e.g.,* videos 109 or 1281). It seems difficult to apply a learning method to rank algorithms on such a configuration. This would also prevent an *x264* user to know whether the configuration 16 will be optimal to get the minimal *bitrate* of video.

Key findings for RQ1.1. Depending on the inputs, the correlations of *encoded sizes* or *bitrate* can be high, close to zero, or even negative: one cannot blindly reuse a configuration prediction model for some performance properties of interest.

RQ1.2 - Are there some configuration options more sensitive to input videos?

To answer this question, we investigate the individual effects of videos on configuration options. These different effects may alter the software performances thus explaining the differences of performance distributions pointed out in Section II-A.

To address this research question, we study importance values of features across different input videos. Three features are strongly influential for a majority of videos: *subme*, *mbtree* and *aq-mode*, but their importance can differ depending on input videos (*e.g.,* for *subme* and video 1365, the importance is 0.83 while for video 40, the importance is only 0.01). Overall, the importance for other features is very low though some videos (*e.g.,* videos 140, 1044 and 1311) introduce noticeable variations: for example, *cabac* is low for the rest of the videos, but represents 78 % of the information for video 1311. We can observe the same results for *bframes* and *badapt* with videos 140 and 1044 too.

Figures 2a and 2b report respectively the option importances (random forest) and the linear regression coefficients. The results detailed in Figure 2a show that for all input videos only few options are important and influential. Coefficients' values are scaled to configurations' *bitrate* and are specific to a video: one cannot directly compare them. However one can rank their strengths w.r.t. other coefficients, or study their signs. The results detailed in Figures 2a and 2b confirm our observations: we can notice that the same three options, *subme*, *mbtree* and *aq-mode*, are highly influential. However, some specific videos can increase the importance of other options. For an example, videos 314 and 378 alter the effects of *subme*; their most influential feature (more than 80 % of the information) is *mbtree*, and they have a Spearman correlation of 0.89 between their distributions. Oppositely, they have low or moderate correlations with the video 1192 (resp. 0.21 for video 314 and 0.42 for video 378), highly sensitive to the feature *subme*.

In Figure 2b, we report the raw coefficients of individual options. Interestingly, *mbtree* can have positive and negative impacts on the *bitrate*. It reveals that setting *mbtree* to true (or false) may have a different effect depending on the input video. Most of the features have positive and negative values; thus, the specific impacts heavily depend on input videos. On the other side, low *subme* values or deactivating *cabac* tend to increase the *bitrate* for most of the videos.

A difference of feature importance between two videos can

change the ranking of performances, thus explaining the low correlations computed in Section II-A.

Key findings for RQ1.2 Three main options *subme*, *mbtree* and *aq-mode* are influential; their high importance is preserved across the majority of the inputs. However, some inputs may decrease their influential strength in favor of other options. Overall, individual options can be sensitive to inputs and explain the variations previously observed.

D. Answering RQ1

> **RQ1 - Do Input Videos Change Performances of x264 Configurations?** Different inputs lead to different configuration rankings and software features' effects; this phenomenon is especially apparent for two performance properties of *x264*. The consequence is that developers should not provide to end-users a unique performance prediction model or a default configuration whatever the input is.

Owing to the significance of the input-sensitivity problem, alternate solutions should be considered.

III. Approach

Since inputs can matter to performance predictions, we propose an approach, called ***Inputec*, able to propose a configuration minimizing a software performance** based on characteristics of an input (here: a video). Figure 3 depicts an overview of *Inputec*. Our approach is driven by both the way *x264* experts/users tune their systems and empirical observations about configurations and inputs. In this section, we explain the principles behind *Inputec*. We then detail the finding of inputs properties for grouping videos by performance, as well as the algorithm for selecting configurations. In the remainder of this section and also in the evaluation, we focus on *bitrate*, since this performance property is the most sensitive to input.

A. Intuition and Principles

Grouping inputs by performance. For mitigating input sensitivity, an idea is to group together inputs based on the performance distributions of configurations. Looking further at our empirical results, we first notice that, for a given video, it is always possible to find another video with very highly correlated *bitrate* distributions. For 95 % of the videos, it is even possible to find another video with a correlation higher than 0.92. To further explore the idea, we extract four performance groups of videos from the dendogram of Figure 1 (1 being on the left, and 4 on the right). We notice that:

- Raw performances vary from one group to another.
- A software feature can be both influential for a given group, and insignificant for another; as an example, *subme* has a lower influence for the second group (value: 0.14) than for groups 1 (value: 0.49) and 3 (value: 0.45).
- *x264* features have different effects depending on the group, but remain stable within the same group. For instance, *mbtree* has various effects on different groups of videos (positive for the first but negative for the second).

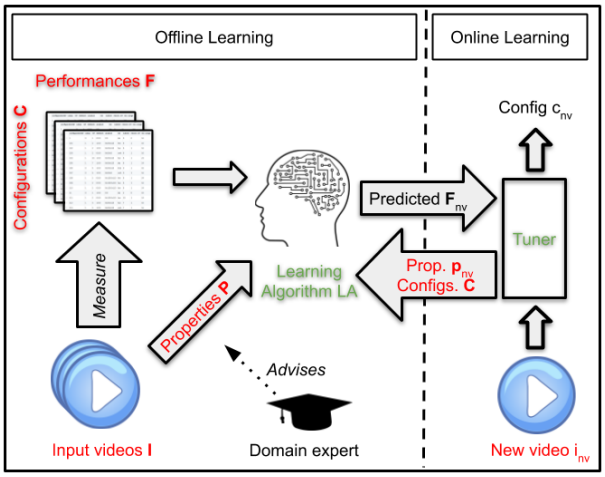


Fig. 3: *Inputec* overall approach

In short, grouping together inputs seems a right approach to reduce input sensitivity. However there is now a problem: we need to map a given input into a group *a priori*, without having access to all measurements.

Tuning knowledge in *x264* The video compression community proposed special algorithms to take the video characteristics into account. Early studies showed evidence that the amount of benefit for each technique varies dramatically for different video scene contents. *x264* provides such algorithms and techniques through options. We further investigated how the *x264* community deals with the problem of tuning configurations, looking at official documentation and online resources (more details about the references in the companion webpage). First, tuning mechanisms: *x264* exposes a preset and tune system. Presets offer pre-defined, default configurations for certain usages with a varying trade-off between encoding time and file size. It is usually combined with the *x264 tune* options that assign additional options that optimize the encoder *for certain types of content*. Specifically, users can specify a few categories of content (film, animation, grain, slideshows) and then *x264* sets the right options' values. Second, *documentation and guidelines*: there are several official and non-official resources that document the impact of each option w.r.t. performance properties and content specificities (*e.g., "scene change detection"*, *"high motion scenes"*, *"HD footage"*).

We can observe that the practice of tuning options' values based on the specifics of the input is clearly advocated.

Unfortunately, the configuration knowledge is mostly written in natural languages and provides high-level, informal recommendations. It is up to experts or users to then select the right values, based on some guesses or tries/errors. Our hypothesis is that what makes a difference between experts and basic users is an implicit knowledge, based on years of experience, that can deal with input sensitivity. The goal of *Inputec* is to mimic *x264* experts' knowledge and fully automate the process through the discovery of inputs' properties that can *a priori* drive the tuning. The domain knowledge is replaced by a model capable

of learning out of examples (configurations' measurements) while the tedious tasks done by the expert decision are replaced by a prediction of the model.

Learning once and for all. Each time an input comes in, a possible approach is to measure dozens of configurations, train a learning model, and eventually tune the system. The cost of measuring can be high, especially if the process should be repeated several times by different users. In contrast, *Inputec* aims to tune without any additional configurations' measurements. The principle is to learn once and for all how inputs' properties related to configurations. *Inputec* then statically tunes a system with no runtime overhead, except the computation of inputs' properties. *Inputec* can be seen as an offline learner that centralizes the configuration knowledge for systematic reuse by end-users.

B. Finding Input Properties

Given a new input fed to *x264* configurable system, *Inputec* should identify the relevant performance group without measuring configurations. It boils down to develop a procedure that computes some properties' values of the input. This procedure **should rely on input properties that are informative enough to discriminate among the groups**; this information is used to train a learning algorithm (see line 14 of Algorithm 1). Furthermore, the properties should be less costly as possible to compute. As a domain knowledge, we use all videos' properties of the Youtube UGC dataset [21]. We then consider two strategies as part of the learning algorithm (LA, see fig. 3): (1) considering all properties; (2) selecting only a subset: we notice that some properties can be computationally expensive and are not that informative for LA. Based on the dendogram of Figure 1, we report on videos' properties that can be used to characterize four performance groups:

- Group 1. action videos (high spatial and chunk complexities, Sports and News);
- Group 2. big resolution videos (low spatial and high temporal complexities, High Dynamic Range);
- Group 3. still image videos (low temporal and chunk complexities, Lectures and HowTo)
- Group 4. standard videos (average properties values, various contents)

The different group characteristics (*i.e.,* inputs' properties) are depicted in Figure 4. For each group (*i.e.,* the columns), we depict the average values of some indicators (*i.e.,* the lines). To describe a bit more the distributions, average values are shown with the standard deviations.

A first observation is that the correlations within each individual group are strong (for Group 4: 0.77) or very strong (for Group 3: 0.87). Within a group, configurations' performance generalize over all videos. Second, we can notice that the performance effects of *x264* features significantly vary across groups. For instance, the feature importance of *mbtree* varies from 0.09 (weak effect on Group 1) to 0.47 (strong effect on Group 2). Moreover, the coefficients of *mbtree* can be either positive or negative: the setting of *mbtree* value (either true or false) can have opposite effects depending of the group.

Group		1. Action	2. Big	3. Still image	4. Standard
Number of Videos		**470**	**219**	**292**	**416**
Video Properties		Spatial ++	Spatial - -	Spatial - -	Width -
		Chunk ++	Temporal ++	Temporal - -	Height -
			Width ++	Chunk - -	Temporal -
Main Categories		Sports	HDR	Lecture	Music
		News		HowTo	Vertical
Feature Importances	mbtree	**0.09** ± 0.09	**0.47** ± 0.2	**0.34** ± 0.22	**0.05** ± 0.07
	aq-mode	**0.27** ± 0.19	**0.13** ± 0.13	**0.04** ± 0.07	**0.15** ± 0.18
Feature Coefficients	mbtree	**0.33** ± 0.19	**-0.68** ± 0.18	**-0.42** ± 0.15	**-0.11** ± 0.15
	aq-mode	**-0.5** ± 0.14	**0.36** ± 0.21	**-0.14** ± 0.14	**-0.29** ± 0.18
Average Correlation		**0.82** ± 0.11	**0.79** ± 0.14	**0.85** ± 0.09	**0.74** ± 0.17
Median Correlation		**0.84**	**0.82**	**0.87**	**0.77**

Fig. 4: Groups of videos based on bitrates distributions

It is exactly why we want to identify groups: as a way to tune options' values depending on input. Finally, there exists video properties and categories to discriminate groups. For instance, Group 1 exhibits a high spatial and chunk complexity while Group 2 exhibits a low spatial and high temporal complexity.

C. Tuning Algorithm

Now we have identified inputs' properties, *Inputec* can use them to select custom configurations. Figure 3 details how *Inputec* works, and Algorithm 1 explains its algorithm. To match our current evaluation, replace LA by a Random Forest regressor, and F with *bitrate*. It consists of two steps.

Offline learning. (lines 1-14) - For each input and each configuration, we consider the lists of input *performances* and properties. We normalize the performances of videos, so the differences between videos do not disturb the learning process; this transformation does not change the rankings of videos. For each input, we prefix each line of configuration options with its input properties. We then aggregate all these information in a dataset (*i.e.,* the predicting variables X), along with the related software performances (*i.e.,* the variable of interest y). Afterwards, the machine learning algorithm associates each set of input properties **and** configuration options (*i.e.,* X) to a performance value (*i.e.,* y), learning how to predict efficiently y with X values.

Online learning. (lines 15-22) - Once the model (or learning algorithm) LA is trained, we feed it with a new set of properties (*i.e.,* coming from a new input video); for each configuration, we use the model to predict the performance of the video. At last, we select the configuration resulting to the minimal value of predicted performances.

IV. Evaluation

This section details the evaluation of the proposed input-aware software tuning, *Inputec*, on the *bitrate* of *x264*'s compression. We define the main research question of this evaluation as follows: **RQ2. Can we use *Inputec* to find configurations adapted to input videos?**

Algorithm 1 - Inputec

1: *// Offline learning*
2: Inputs C configurations, F performances, I input videos, P properties
3: Init X, predicting variables
4: Init Y, variables to predict
5: **for** each input $i \in I$ **do**
6: Read P[i], input i properties
7: **for** each configuration $c \in C$ **do**
8: Add [P[i], c] to X
9: Read F[c, i], configuration c performance for the input i
10: Add F[c, i] to Y
11: **end for**
12: **end for**
13: Init LA, learning algorithm
14: Assign LA = $argmin_f(|f(X) - Y|)$
15: *// Online learning*
16: Input i_{nv}, new video
17: Init $\hat{F_{nv}}$, predicted performances for i_{nv}
18: Read p_{nv} = P[i_{nv}], input i_{nv} properties
19: **for** each configuration $c \in C$ **do**
20: Assign $\hat{F_{nv}}[c]$ = LA(p_{nv}, c])
21: **end for**
22: Output $\hat{c_{nv}} = argmin_{c \in C}(\hat{F_{nv}}[c])$

A. Research Questions

To answer this question, we use the algorithm defined in Section III on the dataset presented in Section II-B. First of all, and to avoid biasing the results, we separate the list of inputs in two parts; training and validation. We then apply the offline learning part of *Inputec* algorithm on the training dataset, and validate it by applying the online learning part of the algorithm on the validation dataset. To evaluate *Inputec*, we compare it to different baselines. Each baseline corresponds to a concrete situation;

Baseline B1 - Model Reuse configuration. We arbitrarily choose a first video, learn a performance model on it, and select the best configuration minimizing the performance for this video. This baseline represents the error made by a model trained on a source input (*i.e.,* a first video) and transposed to a target input (*i.e.,* a second video, different from the first one), without considering the difference of content between the source and the target. In theory, it corresponds to a fixed configuration, optimized for the first video. We add B1 to measure how we can improve the standard performance model with *Inputec*.

Baseline B2 - Preset configuration. We select the configuration having the lowest sum of *bitrate* rankings for the training set of videos, and study this configuration's distribution on the validation set. B2 represents the best compromise we can find, working for all input videos. In terms of software engineering, it acts like a preset configuration proposed by *x264* developers. Beating this configuration shows that our approach chooses a custom configuration, tailored for the input characteristics.

Baseline B3 - Average configuration. This baseline computes the average performance of configurations for each video of the validation dataset. It acts as a witness group, reproducing the behavior of a non-expert user that experiments *x264* for the first time, and selects uniformly one of the 201 configurations of our dataset.

Baseline B4 - Best configuration. Similarly, we select the best configuration (*i.e.,* leading to the minimal performance for the set of configurations). We consider this configuration as the upper limit of the potential gain of performance; since our approach chooses a configuration in the set of 201 possible choices, we can not beat the best one of the set; it just shows how far we are from the best performance value. Otherwise, either our method is not efficient enough to capture the characteristics of each video, or the input sensitivity does not represent a problem, showing that we can use an appropriate but fixed configuration to optimize the performances for all input videos.

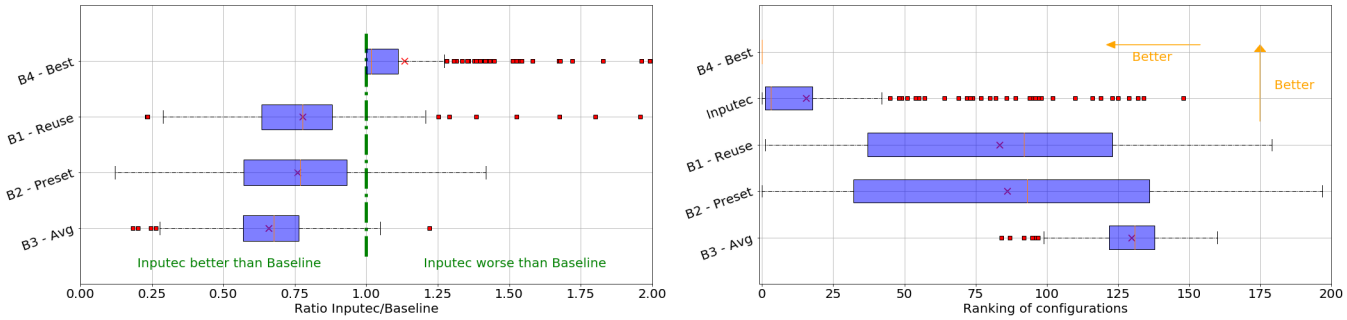
The software performances cannot be compared between videos having different *bitrate* scales. For this reason, to compare *Inputec* and the baselines, we decided to consider the two following **metrics**:

1) The **ranking** of *Inputec* and the baseline's configuration performances. The more efficient the method, the lower the ranking.
2) The distributions of **ratios** between the general *Inputec* method and the rest of the different baselines. The ratios of *Inputec* over the baseline performances prediction should be lower than 1 if and only if *Inputec* is better than the baseline (because it provides a lower *bitrate* than the baseline). As an example, for a ratio of $0.6 = 1 - 0.4 = 1 - \frac{40}{100}$, we gain $40\,\%$ of *bitrate* with our method compared to the baseline. Oppositely, we loose $7\,\%$ of *bitrate* with our method compared to the baseline for a ratio of 1.07.

B. Results

We present results for the following research questions w.r.t. the methodology defined in Section IV-A. We depict the ratios distributions (between *Inputec* and the baselines) in Figure 5a and performance rankings for all methods and baselines in Figure 5b. Baselines B1 and B2 are fixed configurations, they have naturally the highest interquartile ranges (respectively 86 and 104 for ranks). For several videos (*e.g.,* video 1052 for B1 or 293 for B2), the fixed configurations outperform the near-optimal configuration selected by *Inputec*. In general, however, our approach yields to a median reduction greater than $20\,\%$ of the *bitrate* compared to both baselines.

Apart from few configurations having a high rank (between 50 and 146), the configurations selected are near-optimal to minimize the *bitrate*; **half the configurations proposed by *Inputec* are in the top-5 ranking, and more than $75\,\%$ in the top-25 ranking**. Three Welch's t-tests [23] confirm that the rankings of *Inputec* are significantly different from B1, B2 and B3 rankings. Besides, *Inputec* can diminish the *bitrate* by $33\,\%$ for a user selecting randomly one of the 201 proposed.



(a) Performance ratios *Inputec*/Baseline

(b) Configuration rankings

Fig. 5: Evaluation of *Inputec*

Our solution gives a configuration representing less than 105 % of the best configuration median performance (*i.e.,* baseline B4), which is close to the best value. But some videos are not handled correctly by *Inputec*, and results in larger ratios (mostly between 1 and 2).

Effects of measurement costs on *Inputec* To reduce the *offline* cost of *Inputec*, we trained it with a reduced number of configurations (*i.e.,* using 20 configuration per video instead of 201). This variant leads to a decrease of performances (median ranking: 36.5 against 5 for *Inputec*), but still outperforms the baselines B1, B2 and B3.

Effects of properties' selection on *Inputec* To reduce the *online* cost of *Inputec* (*i.e.,* the time needed to calculate the input properties for a new video fed to *Inputec*), we only kept the computationally affordable properties in the performance model; this variant leads to the same results as *Inputec* (same median, average ranking: 16.4 against 15.3 for *Inputec*). It is the best compromise we could find between accuracy and computation time of input properties.

C. Answering RQ2

RQ2 - Can we use *Inputec* to find configurations adapted to input videos? *Inputec* selects configurations tailored for inputs, outperforms other baselines in terms of performances, and provides a near-optimal configuration for a vast majority of input videos. In average, a non-expert can diminish the *bitrate* by 33 % if she uses *Inputec*, finding a configuration in the top-5 ranking half of the time.

Replication of the study. We did observe similar results for the *encoded sizes* of videos. It is a performance property also significantly impacted by the input sensitivity problem. More details can be found in the companion repository

V. Discussion

A. Inputec *for centralizing configuration knowledge and shifting individuals' costs*

A key idea of *Inputec* is to invest resources for capturing a configuration knowledge that generalizes to any input. This approach is in sharp opposition with *online* learning methods that measure configurations to then predict a configuration. The learning of *Inputec* occurs in an offline phase and then pays off at every usage of the configurable system. We argue that *Inputec* is particularly suited for *x264*. The huge diversity of inputs (videos) makes the problem of configuration changes exacerbated. If every time a new entry is processed a user has to run dozens of configurations, the overall cost would be huge. The expected benefit of *learning once and for all* is to reduce the individual cost of tuning as much as possible with a strategy that does not require expensive calculations. An organization would need to accept this cost and invest resources to benefit the user ecosystem.

Runtime overhead. Our empirical results show that it is possible to select inputs' properties that are cheap to compute. For *x264*, the cost of *Inputec* is almost zero and thus outperforms online learning approaches that would require measurements of several configurations. We do not claim, though, that the application of *Inputec* in another domain will necessarily lead to the finding of cheap inputs' properties; it is actually an open question (see Section III-B).

Cost of training *Inputec*. The cost of *Inputec* in the offline phase can arguably be important. We have shown that *Inputec* does not need to measure all configurations: only a small sample (20 %) can be used to achieve competing results. Specifically, we estimate that the computational cost of *Inputec* corresponds to the use of a single machine during one week. This cost seems affordable for an organization or a community like *Inputec* that heavily relies on benchmarks and continuous integration [24], [25]. The measurements can also be distributed among different partners and contributors of the project. We also believe there is room to further reduce the cost through the consideration of a smaller sample of videos (instead of 1397 videos as we did); we leave it as future work. Overall, the decision to invest resources for capturing configuration knowledge is a trade-off between the cost and accuracy of *Inputec* and the benefits of tuning for the ecosystem.

Offline vs transfer learning. Online learning can be realized through traditional learning and configurations' measurements

for each video. To reduce the runtime cost of measurements, another possibility is to rely on transfer learning techniques (model shifting [14], Gaussian processes [15], Learning to Sample [16], [26], *etc.*). The approach would be to transfer an existing prediction model of a given video (source) onto another video (target). However, such approaches do require configurations' measurements to learn the transfer function that maps the source onto the target. Though, the size of an effective training set may be reduced compared to traditional learning, transfer learning suffers from the same problem as online learning techniques: the cost is deferred to individuals each time an input comes in. Moreover, transfer learning is heavily dependent on the availability of measurements for selecting the right sources. It is similar to *Inputec*: there is a necessary phase to gather measurements that would be further reused. In fact, *Inputec* can be seen as an offline transfer technique without online learning and the need to measure configurations. An unexplored research direction is to use transfer learning in the offline phase of *Inputec*; we leave it as future work.

B. *Can* Inputec *be applied to other configurable systems?*

Inputec should be applicable to any configurable system under the conditions that: (1) there is a sensitivity to inputs: if there is no input or no sensitivity, then there is no problem; (2) one can find inputs' properties that are informative and for which the runtime cost of their computation is not prohibitive.

Examples of such systems naturally include alternatives to *x264* (*e.g.,* x265, libvpx) or video processing tools (*e.g.,* ffmpeg). But *Inputec* is not limited to videos' inputs. For instance, the performance of satisfiability (SAT) solvers heavily depends on SAT formulae. SATzilla shows that features of SAT formulas can be used to select the best solver among existing ones, achieving state-of-the-art results. Hence *Inputec* could well be used in this context, since SAT solvers are highly configurable. In contrast to SATzilla, *Inputec* would select the best configuration of a given configurable solver - not the best solver among candidates (see also Section VII for a discussion about the algorithm selection problem).

Database management systems (*e.g.,* SQlite, Postgres, MySQL) are configurable systems subject to performance variations depending on specific workloads (see *e.g.,* [27]). Compilers (*e.g.,* gcc, clang) come with different options that are sensitive to input programs (see *e.g.,* [28]). For such kinds of systems, it remains a problem to capture properties of workloads or programs that can systematically be used to tune configurations. There are some positive experiences though, like the compiler PetaBricks [29]. Our hypothesis is that some application domains may be more challenging to learn once and for all; we call to further investigate this empirical question.

C. *Implications for practitioners and researchers in configurable systems.*

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. A learning algorithm, a sampling strategy, or a transfer technique may well be highly accurate for an input and still inaccurate for others. The scientific community should be extremely careful with this input sensitivity problem.

Another consequence of our results is that practitioners should measure the impacts of inputs on configurations and react accordingly, which may take different forms: (1) default configurations that neglect inputs' specifics should be reconsidered, possibly by replacing them with an offline, input-aware tuner like *Inputec*; (2) the documentation should be as clear as possible to inform or even guide users; (3) for testing software performances (*e.g.,* ensuring non-regression in a continuous integration system), a set of representative inputs should be used. More research is definitely needed in such software engineering contexts.

VI. Threats to Validity

Construct validity. Due to budget constraints, we did not include all the features of *x264* in the experimental protocol. In particular *crf* could be influential for the *bitrate*. However, we consider a representative range of 24 features, altered by commonly used configurations defined by the *x264* community (*i.e.,* presets).

Internal Validity. The first threat to internal validity is the selection of the learning algorithm for our study and its parameter settings. To alleviate this issue, we compare three different methods (linear regressions, random forests regressors and neural networks) and selects the best one. Another threat to validity is related to the machine learning prediction: one cannot be sure to retrieve the same results between different launches. To mitigate this issue, for *Inputec*, we replicate the experiments five times and present the average results. Our results can be subject to measurement bias; we alleviate this threats by making sure only our experiment was running on the server we used to measure *x264* performances. It has several benefits: we can guarantee we use similar hardware (both in terms of CPU and disk); 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 input videos on local folders. Because of the amount of resources needed to compute all the videos, we did not repeat the measurement process for several input videos. Based on the previous results of Pereira *et al.* [30], we assume that videos often maintain stable performances between different launches of the same configuration.

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 one interesting performance property, the results may be subject to this system and property. However, we focused in a single system to be able at making robust and reliable statements about whether a machine-learning approach can be used in such a scenario. Once we are able to prove that, we can then perform such an analysis also over other systems and properties to generalize the results. Our experiment is designed to measure the influence of input videos on *x264* performance; as pointed out in [7], there is always a risk to only

optimize software performances for the chosen benchmark. To mitigate this risk, we select the Youtube UGC dataset [21], a huge dataset of 1397 videos covering a wide range of different input particularities and provided consistent results across the experiments.

VII. Related Work

Previous studies on video compression. Netflix conducts a large-scale study for comparing the compression performance of *x264*, *x265*, and *libvpx* [13]. 5000 12-second clips from Netflix catalog were used covering a wide range of genres and signal characteristics. However, only two configurations were considered and the focus of the study was not on predicting performances. Maxiaguine *et al.* [12] classify multimedia streams in groups depending on their characteristics and study their performances (*i.e.,* multimedia processing time and I/O rate) on MPSoC platforms. Pereira *et al.* [30] study the effect of sampling on *x264* configuration performance models for 19 input videos on two performance properties. Our study covers much more inputs and quantitatively characterizes input sensitivity (RQ1). We also propose a method to deal with input sensitivity (RQ2). Valov *et al.* [31] proposed a method to transfer the Pareto frontiers (encoding time and size) of *x264* performances across heterogeneous hardware environments. The inputs (video) remain fix however, which is an immediate threat to validity. In fact this threat is shared by numerous studies on configurable systems that consider *x264* configurations with the same video input (see [32] for the references).

Machine learning and configurable systems. Machine learning techniques have been widely considered in the literature to learn software configuration spaces [9], [14], [16], [26], [32]–[37]. Several works have proposed to predict performance of configurations, with several use-cases in mind for developers and users of configurable systems: the maintenance and interpretability of configuration spaces [10], the selection of an optimal configuration [9], [37], the automated specialization of configurable systems [11], [38]. 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. In response, transfer learning techniques [14], [16], [26], [34]–[36] have been proposed, mostly to deal with a change in the computing environments (see discussions in section V-A).

Input sensitivity. There are numerous work addressing the performance analysis of software systems [39]–[43] depending on different workload contents. However, existing empirical 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. It may limit some key insights about the input sensitivity problem. In response, we perform an in-depth and controlled study of the same system to make it vary in the large, both in terms of configurations and inputs.

Selection problem. The automated algorithm selection problem is subject to intensive research [44]–[47]: given a computational problem, a set of algorithms, and a specific problem instance to be solved, the problem is to determine which of the algorithms can be selected to perform best on that instance. In our case the set comprises all (valid) configurations of a single, parameterized algorithm (whereas the set of algorithms come from different software implementation and systems for the problem of algorithm selection). As stated in [44] (Section 6), it is an open problem because (1) the space of valid configurations to select from is typically very large; (2) learning the mapping from instance features (*i.e.,* inputs' properties) to configurations is challenging. We addressed this problem with *Inputec* and showed the existence of a configurable system for which the approach is effective. In the context of the compiler PetaBricks, Ding *et al.* [29] proposed a technique for auto-tuning based on features of programs. The approach relies on a clustering method to automatically group related programs. In contrast, *Inputec* relies on supervised learning to identify informative inputs' properties.

VIII. Conclusion

We conducted a large study over 1300+ inputs fed to 200+ software configurations that shows the significance of the input sensitivity problem on 5 performance properties of the *x264* configurable system. To overcome this neglected problem, we defined *Inputec*, an input-aware software tuner. Without any measurement, *Inputec* automatically selects a close to optimal configuration for an input fed to a software. We deliver two main conclusions:

- *inputs matter as (much as) configuration options*: ignoring this lesson leads to the learning of inaccurate performance prediction models and ineffective recommendations for developers and end-users;
- *capitalizing the configuration knowledge once and for all* is possible: instead of measuring new configurations each time an input is fed to a configurable system, one can identify input properties that are then used to select a custom configuration.

Our empirical results have implications for practitioners and researchers in configurable systems, and call for further research and replicating our study.

- To what extent inputs fed to a configurable system impact performance distributions? May other configurable systems be more or less sensitive to inputs, depending on quantitative properties of interest?
- Is it possible to find input properties that *Inputec* can exploit? We did find actionable properties in the context of videos and *x264*, but other kinds of inputs may be challenging to tackle.

Another research direction is to reduce the cost of *Inputec* in the offline learning phase, when measurements are gathered.

The questions raised in our case study should be considered in different software engineering contexts. Highly configurable systems like compilers, big data tools, solvers, compression tools, or operating systems to name a few are good subject candidates.

REFERENCES

[1] P. Jamshidi and G. Casale, "An uncertainty-aware approach to optimal configuration of stream processing systems," *CoRR*, vol. abs/1606.06543, 2016. [Online]. Available: http://arxiv.org/abs/1606.06543

[2] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-aware performance prediction: A statistical learning approach," in *ASE*, 2013. [Online]. Available: https://dl.acm.org/doi/10.1109/ASE.2013.6693089

[3] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov, "Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption," *Inf. Softw. Technol.*, 2013. [Online]. Available: http://www.cs.cmu.edu/~ckaestne/pdf/IST12.pdf

[4] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki, "Cost-efficient sampling for performance prediction of configurable systems (t)," in *ASE'15*, 2015. [Online]. Available: https://dl.acm.org/doi/abs/10.1109/ASE.2015.45

[5] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki, "Performance prediction of configurable software systems by fourier learning (T)," in *ASE'15*, 2015. [Online]. Available: https://dl.acm.org/doi/10.1109/ASE.2015.15

[6] P. Valov, J. Guo, and K. Czarnecki, "Empirical comparison of regression methods for variability-aware performance prediction," in *SPLC'15*, 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2791060.2791069

[7] A. Alourani, M. Bikas, and M. Grechanik, "Input-sensitive profiling," in *Advances in Computers*. Elsevier, 2016, pp. 31–52. [Online]. Available: https://doi.org/10.1016/bs.adcom.2016.04.002

[8] G. J. Sullivan and T. Wiegand, "Video compression - from concepts to the h.264/avc standard," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.3827&rep=rep1&type=pdf

[9] J. Oh, D. Batory, M. Myers, and N. Siegmund, "Finding near-optimal configurations in product lines by random sampling," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 61–71. [Online]. Available: https://doi.org/10.1145/3106237.3106273

[10] N. Siegmund, A. Grebhahn, C. Kästner, and S. Apel, "Performance-influence models for highly configurable systems," in *ESEC/FSE'15*, 2015. [Online]. Available: https://doi.org/10.1145/2786805.2786845

[11] P. Temple, M. Acher, J.-M. A. Jézéquel, L. A. Noel-Baron, and J. A. Galindo, "Learning-based performance specialization of configurable systems," IRISA, Inria Rennes ; University of Rennes 1, Research Report, feb 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01467299

[12] A. Maxiaguine, Yanhong Liu, S. Chakraborty, and Wei Tsang Ooi, "Identifying "representative" workloads in designing mpsoc platforms for media processing," in *2nd Workshop onEmbedded Systems for Real-Time Multimedia, 2004. ESTImedia 2004.*, 2004, pp. 41–46. [Online]. Available: https://ieeexplore.ieee.org/document/1359702

[13] Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne Aaron, "A Large-Scale Comparison of x264, x265, and libvpx – a Sneak Peek," netflix-study-link, 2016.

[14] P. Valov, J. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki, "Transferring performance prediction models across different hardware platforms," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*, 2017, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/3030207.3030216

[15] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer learning for improving model predictions in highly configurable software," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Los Alamitos, CA: IEEE Computer Society, 5 2017, pp. 31–41. [Online]. Available: https://arxiv.org/abs/1704.00234

[16] P. Jamshidi, M. Velez, C. Kästner, and N. Siegmund, "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, 2018, pp. 71–82. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3236024.3236074

[17] M. G. Kendall, *Rank correlation methods*. Griffin, 1948.

[18] J. D. Evans, *Straightforward statistics for the behavioral sciences*. Thomson Brooks/Cole Publishing Co, 1996.

[19] J. Dorn, S. Apel, and N. Siegmund, "Generating attributed variability models for transfer learning," in *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, ser. VAMOS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3377024.3377040

[20] W. R. Zwick and W. F. Velicer, "Factors influencing four rules for determining the number of components to retain," *Multivariate Behavioral Research*, vol. 17, no. 2, pp. 253–269, 1982, pMID: 26810950. [Online]. Available: https://doi.org/10.1207/s15327906mbr1702_5

[21] Y. Wang, S. Inguva, and B. Adsumilli, "YouTube UGC dataset for video compression research," in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, Sep. 2019. [Online]. Available: https://doi.org/10.1109/mmsp.2019.8901772

[22] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[23] D. W. Zimmerman, "A note on preliminary tests of equality of variances," *British Journal of Mathematical and Statistical Psychology*, vol. 57, no. 1, pp. 173–181, 2004.

[24] https://nightlies.videolan.org/, "VLC media player nightly builds," August 2020.

[25] http://fatebeta.ffmpeg.org, "FFmpeg Automated Testing Environment," August 2020.

[26] P. Jamshidi, N. Siegmund, M. Velez, C. Kastner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Oct. 2017. [Online]. Available: https://dl.acm.org/doi/10.5555/3155562.3155625

[27] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1009–1024. [Online]. Available: https://doi.org/10.1145/3035918.3064029

[28] A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, "A survey on compiler autotuning using machine learning," *ACM Comput. Surv.*, vol. 51, no. 5, Sep. 2018. [Online]. Available: https://doi.org/10.1145/3197978

[29] Y. Ding, J. Ansel, K. Veeramachaneni, X. Shen, U.-M. O'Reilly, and S. Amarasinghe, "Autotuning algorithmic choice for input sensitivity," in *ACM SIGPLAN Notices*, vol. 50, no. 6. ACM, 2015, pp. 379–390. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2813885.2737969

[30] J. Alves Pereira, M. Acher, H. Martin, and J.-M. Jézéquel, "Sampling effect on performance prediction of configurable systems: A case study," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 277–288. [Online]. Available: https://doi.org/10.1145/3358960.3379137

[31] P. Valov, J. Guo, and K. Czarnecki, "Transferring pareto frontiers across heterogeneous hardware environments," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 12–23. [Online]. Available: https://doi.org/10.1145/3358960.3379127

[32] J. A. Pereira, H. Martin, M. Acher, J.-M. Jézéquel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," *ArXiv*, vol. abs/1906.03018, 2019. [Online]. Available: https://arxiv.org/abs/1906.03018

[33] C. Quinton, M. Vierhauser, R. Rabiser, L. Baresi, P. Grünbacher, and C. Schuhmayer, "Evolution in dynamic software product lines," *Journal of Software: Evolution and Process*, p. e2293, 2020.

[34] V. Nair, R. Krishna, T. Menzies, and P. Jamshidi, "Transfer learning with bellwethers to find good configurations," *CoRR*, vol. abs/1803.03900, 2018. [Online]. Available: http://arxiv.org/abs/1803.03900

[35] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Using bad learners to find good configurations," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, 2017, pp. 257–267. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106238

[36] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," *IEEE Transactions on Software Engineering*, 2018.

[37] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 49–60. [Online]. Available: https://doi.org/10.1145/3106237.3106256

[38] P. Temple, M. Acher, J.-M. Jezequel, and O. Barais, "Learning contextual-variability models," *IEEE Software*, vol. 34, no. 6, pp. 64–70, Nov. 2017. [Online]. Available: https://doi.org/10.1109/ms.2017.4121211

[39] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance analysis of private blockchain platforms in varying workloads," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, Jul. 2017. [Online]. Available: https://doi.org/10.1109/icccn.2017.8038517

[40] E. Coppa, C. Demetrescu, I. Finocchi, and R. Marotta, "Estimating the Empirical Cost Function of Routines with Dynamic Workloads," in *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '14. New York, NY, USA: ACM, 2014, pp. 230:230–230:239. [Online]. Available: http://doi.acm.org/10.1145/2581122.2544143

[41] H. FathyAtlam, G. Attiya, and N. El-Fishawy, "Comparative study on CBIR based on color feature," *International Journal of Computer Applications*, vol. 78, no. 16, pp. 9–15, Sep. 2013. [Online]. Available: https://doi.org/10.5120/13605-1387

[42] S. F. Goldsmith, A. S. Aiken, and D. S. Wilkerson, "Measuring empirical computational complexity," in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC-FSE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 395–404. [Online]. Available: https://doi.org/10.1145/1287624.1287681

[43] P. Leitner and J. Cito, "Patterns in the chaos—a study of performance variation and predictability in public iaas clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, Apr. 2016. [Online]. Available: https://doi.org/10.1145/2885497

[44] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary Computation*, vol. 27, no. 1, pp. 3–45, 2019. [Online]. Available: https://doi.org/10.1162/evco_a_00242

[45] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523. [Online]. Available: https://dl.acm.org/doi/10.1007/978-3-642-25566-3_40

[46] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: portfolio-based algorithm selection for sat," *Journal of artificial intelligence research*, vol. 32, pp. 565–606, 2008. [Online]. Available: https://www.aaai.org/Papers/JAIR/Vol32/JAIR-3214.pdf

[47] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "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, 2013, pp. 847–855. [Online]. Available: https://doi.org/10.1145/2487575.2487629