



**HAL**  
open science

## Learning input-aware performance models of configurable systems: An empirical evaluation

Luc Lesoil, Helge Spieker, Arnaud Gotlieb, Mathieu Acher, Paul Temple, Arnaud Blouin, Jean-Marc Jézéquel

► **To cite this version:**

Luc Lesoil, Helge Spieker, Arnaud Gotlieb, Mathieu Acher, Paul Temple, et al.. Learning input-aware performance models of configurable systems: An empirical evaluation. *Journal of Systems and Software*, 2023, pp.111883. 10.1016/j.jss.2023.111883 . hal-04271476

**HAL Id: hal-04271476**

**<https://hal.science/hal-04271476v1>**

Submitted on 6 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Learning Input-aware Performance Models of Configurable Systems: An Empirical Evaluation

Luc Lesoil<sup>a</sup>, Helge Spieker<sup>b</sup>, Arnaud Gotlieb<sup>b</sup>, Mathieu Acher<sup>a,\*</sup>, Paul Temple<sup>a</sup>, Arnaud Blouin<sup>a</sup>, Jean-Marc Jézéquel<sup>a</sup>

<sup>a</sup>*Univ Rennes, Inria, INSA Rennes, CNRS, IRISA, France*

<sup>b</sup>*Simula Research Laboratory, Oslo, Norway*

---

## Abstract

Modern software-based systems are highly configurable and come with a number of configuration options that impact the performance of the systems. However, selecting inappropriate values for these options can cause long response time, high CPU load, downtime, RAM exhaustion, resulting in performance degradation and poor software reliability. Consequently, considerable effort has been carried out to predict key performance metrics (execution time, program size, energy consumption, etc.) from the user's choice of configuration options values. The selection of inputs (e.g., *JavaScript* scripts embedded in a web page interpreted by *Node.js* or input videos encoded with *x264* by a streaming platform) also impacts software performance, and there is a complex interplay between inputs and configurations. Unfortunately, owing to the huge variety of existing inputs, it is yet challenging to automate the prediction of software performance whatever their configuration and input. In this article, we empirically evaluate how supervised and transfer learning methods can be leveraged to efficiently learn performance models based on configuration options and input data. Our study over 1,941,075 data points empirically shows that measuring the performance of configurations on multiple inputs allows one to reuse this knowledge and train performance models robust to the change of input data. To the best of our knowledge, this is the first domain-agnostic empirical evaluation of machine learning methods addressing the input-aware performance prediction problem.

*Keywords:* Performance Prediction, Software Variability, Input Sensitivity, Configurable Systems, Learning Models

---

## 1. Introduction

Most modern software systems are widely configurable, featuring many configuration options that users can set or modify according to their needs, for

---

\*Corresponding author

*Email address:* [mathieu.acher@irisa.fr](mailto:mathieu.acher@irisa.fr) (Mathieu Acher)

4 instance, to maximise some performance metrics (*e.g.*, execution time, energy  
5 consumption). As a software system matures, it diversifies its user base and  
6 adds new features to satisfy new needs, increasing its overall number of options.  
7 However, manually quantifying the individual impact of each option and their in-  
8 teractions quickly becomes tedious, costly and time-consuming, which reinforces  
9 the need to automate how to study and combine these options together. Soft-  
10 ware reliability can be largely degraded if inappropriate configuration options  
11 are selected or if ageing-related bugs such as configuration-dependent memory  
12 leaks remain undetected [74].

13 To address these issues, researchers typically apply *machine learning (ML)*  
14 techniques [21, 63] to learn performance models from the selection of configu-  
15 ration options and/or software modules. Conversely, with an accurate perfor-  
16 mance predictive model, it becomes possible to predict the performance of any  
17 configuration, to find an optimal configuration, or to identify, debug, and reason  
18 about influential options of a system [52, 26, 66, 21, 44, 45, 60, 61, 30]. A recent  
19 survey [52] synthesized the large effort conducted in the software engineering,  
20 software variability, and software product line engineering communities.

21 However, the performance variability of a given software system also obvi-  
22 ously depends on its input data [2, 72, 38, 77, 8], *e.g.*, a video compressed by a  
23 video encoder [38] such as *x264*, a program analyzed by a compiler [8, 11] such  
24 as *gcc*, a database queried by a DBMS [77] such as *SQLite*. All these kinds of  
25 inputs might interact with the configuration space of the software [11, 3, 42, 34].  
26 For instance, an input video with fixed and high-resolution images fed to *x264*  
27 could reach high compression ratios if configuration options like *mbtree* are ac-  
28 tivated. The same option will not be suited for a low-resolution video depicting  
29 an action scene [38] with lots of changes among the different pictures, leading  
30 to different performance distributions, as for input videos in Figure 1<sup>1</sup>. The  
31 interplay between input data and configurations has not caught a great atten-  
32 tion in the literature [52]. It is a threat of practical interests, since performance  
33 models of configurable systems or software product lines, can be inaccurate and  
34 deprecated whenever a new input data is processed.

35 Recent works empirically show the significance of inputs (also called work-  
36 loads) when predicting the performance of configurable systems and software  
37 product lines. Pereira *et al.* [3] showed that the performance distribution of  
38 1152 configurations of *x264* heavily depends on the inputs (19 videos) processed.  
39 Practically, a good configuration can be a bad one depending on the processed  
40 video; some configuration options have varying influence and importance de-  
41 pending on videos; a performance prediction model can be inaccurate if blindly  
42 reused whatever the video. Recent empirical results of Mühlbauer *et al.* [42]  
43 and Lesoil *et al.* [34] over different software systems, configurations and inputs  
44 demonstrate that inputs can induce substantial performance variations and in-  
45 teract with configuration options, often in non-monotonous ways. As a result,

---

<sup>1</sup>Videos 1 and 2 are extracted from our dataset (Animation\_1080P-5083 and Anima-  
tion\_1080P-646f)

46 inputs should be considered when building performance prediction models to  
 47 maintain and improve representativeness and reliability.

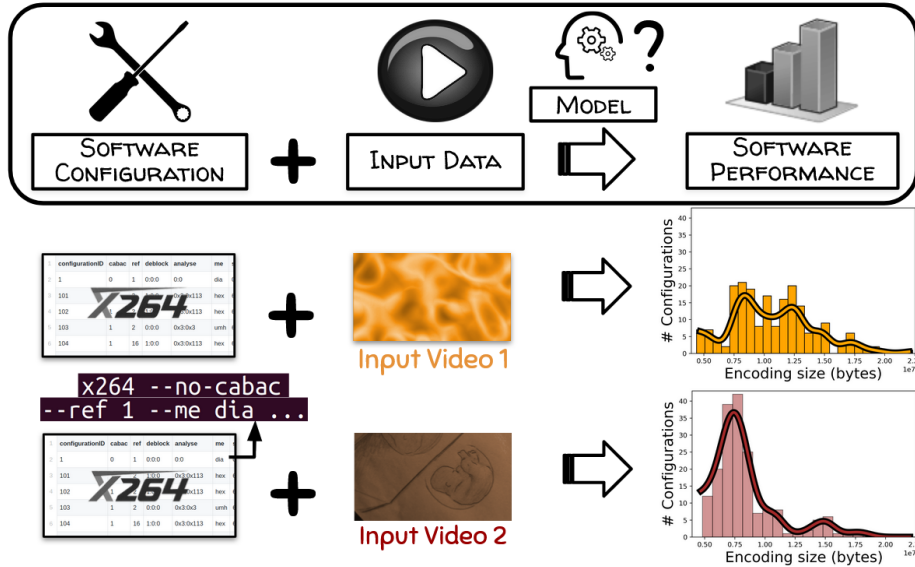


Figure 1: The *performance prediction problem*: how to predict software performance considering both configurations and inputs?

48 Measuring all configurations for all possible inputs of a configurable system  
 49 is the most obvious path to resolve the issue. Given a potentially infinite input  
 50 space, it is however either too costly and infeasible in practice or impossible.  
 51 ML techniques are usually employed to measure only a sample of configurations  
 52 and then use these configurations’ measurements to build a performance model  
 53 capable of predicting the performance of other configurations (i.e., configura-  
 54 tions not measured before). However, these measurements are obtained on a  
 55 specific input and are already costly to compute. Systematically repeating this  
 56 process for many inputs would explode the budgets of end-users and organiza-  
 57 tions. The inputs add a new dimension to the problem of learning performance  
 58 models. The combined problem space (configuration and input dimension) re-  
 59 quires substantially more observations and measurements. It further increases  
 60 the computational cost, since it requires running configurations over many input  
 61 samples. The available budget end-users can dedicate to the measurements of  
 62 configurations over inputs is limited by construction. *Hence, the challenge is to*  
 63 *learn an accurate performance model, aware of configurations and inputs, with*  
 64 *the lowest budget.*

65 Several input-aware approaches can be envisioned. The first is to learn from  
 66 scratch a performance model, whenever an input is fed to a configurable system.  
 67 Many works target the scenario where users build their own performance  
 68 models for their own inputs and workloads [21, 60, 52, 3]. Unfortunately, users  
 69 need to measure a sample of configurations for building a new prediction model  
 70 each time a new input should be processed. The computational cost can be

71 prohibitive as it occurs in an *online* setting (at runtime). There is no reuse  
72 of past observations, knowledge, and performance models. Another second ap-  
73 proach, at the opposite of the spectrum, is to pre-train in an *offline* setting  
74 a set of performance models for different inputs and configurations (as, *e.g.*,  
75 proposed in [11]). The upfront cost can be high but can pay off since these  
76 models are systematically reused when a new input comes in. In the example  
77 of a configurable video encoder, performance models could be learned offline  
78 and reused each time a new video (input) is fed. The advantage is that users  
79 typically have only a small budget and cannot make additional measurements at  
80 run-time. The counter-part is that pre-trained models can have forgotten some  
81 inputs and be (much) less accurate than a performance model trained specifi-  
82 cally on an input. At least, it is a hypothesis worth studying. In between, a  
83 third approach is to use both online and offline settings through transfer learn-  
84 ing [52, 26, 4, 37, 68]. Certain transfer learning methods were originally intended  
85 to handle changes in computing environments, not actual inputs’ changes, ne-  
86 cessitating the development of new techniques that could effectively leverage  
87 specific input characteristics [25, 68, 26, 43, 67, 33, 24]. The principle is to  
88 adapt existing performance models (pre-trained in an offline setting over mul-  
89 tiple inputs and configurations) thanks to additional measurements gathered  
90 over a specific input to process. The hope is to transfer the model with very  
91 few measurements at run time.

92 In this article, we study and compare the cost-effectiveness of these three  
93 input-aware approaches over a large dataset comprising 8 software systems,  
94 hundreds of configurations and inputs, and dozens of performance properties,  
95 spanning a total of 1,941,075 configurations’ measurements. The distinction  
96 between offline and online learning has not received much attention yet (*e.g.*,  
97 most works only apply either ”offline” or ”online” learning), certainly because of  
98 the lack of consideration of input data that can significantly alter the accuracy  
99 of predictive performance models of configurations, as is empirically shown in  
100 the rest of the article. We also consider the peculiarities of inputs (*i.e.*, their  
101 properties) to guide the transfer or reuse of pre-trained models. To the best  
102 of our knowledge, our work is the first domain-agnostic empirical evaluation  
103 of ML methods addressing the input-aware performance prediction problem in  
104 both online and offline settings.

105 Our contributions are as follows:

- 106 1. We perform an extended *comparative study* of performance model training  
107 approaches, including supervised learning as well as transfer learning. We  
108 also present their costs and error levels when addressing the performance  
109 prediction problem;
- 110 2. We provide guidelines to the user who faces a performance prediction problem  
111 and propose a learning-based solution based on her constraints and resources,  
112 *i.e.*, based on a trade-off between costs in offline and online settings.
- 113 3. We publish the code, the dataset, and the results of this paper online<sup>2</sup> as a

---

<sup>2</sup>See the companion repository at <https://github.com/simula-vias/>

114 basis for future work on predictive performance models.

## 115 2. Problem

116 As measuring the performance of each software configuration and input prop-  
117 erties takes time and is computationally costly, we define three different scenar-  
118 ios based on user personas that have distinct levels of resources (*i.e.*, time and  
119 computation power). These scenarios lead to different learning strategies: a  
120 model pre-trained on multiple inputs used only as-is (*offline learning*); a model  
121 trained on a single input whenever an end-user processes an input (*supervised*  
122 *online learning*); a model pre-trained on multiple inputs but that will be adapted  
123 via *transfer learning* by an end-user. These three learning strategies have pros  
124 and cons and are further evaluated in the rest of the article (see Section 4).

### 125 2.1. User Persona (UP)

126 Performance prediction of software configurations has several interests for  
127 organizations, individual users and developers of configurable systems [21, 52,  
128 25, 26, 68, 67]:

- 129 • prediction is interesting per se since users know the performance value  
130 they will get and this quantitative information is actionable. It allows  
131 users to determine whether it reaches a certain limit (or is within accept-  
132 able boundaries) and take informed decisions. For instance, users could  
133 know that the configuration *c1* of video encoding will be 56 s (less than  
134 1 min, acceptable) while the configuration *c2* is 589 s (more than 10 min,  
135 unacceptable).
- 136 • prediction can help explore tradeoffs by (de)activating some options and  
137 see the concrete, quantitative effect on performance. Users are usually  
138 not optimizing w.r.t. one dimension (*e.g.*, execution time) but have also  
139 technical constraints related to output quality (*e.g.*, users do not want  
140 to alter much video quality and avoid using *mbtree* in *x264*) or other  
141 metrics in mind (*e.g.*, size of the output). Developers can also explore the  
142 configuration space to pinpoint inefficient performance and hopefully of  
143 some configurations;
- 144 • interpretable information can be extracted out of prediction models, like  
145 feature importance or interactions across features [40, 60, 12]. This infor-  
146 mation is useful for users and developers in charge of configuring, main-  
147 taining or debugging configurable software systems.

148 In these three cases, inputs can dramatically alter performance distributions  
149 and thus lead to inaccurate performance prediction if the specificities of input  
150 are not taken into account [41, 34]. Hence, UP can adopt different strategies to

151 adapt the performance model, with different computational costs and quality of  
152 performance predictions.

153 Let us consider UP  $\mathcal{A}$  which is in a rush and has to quickly deploy configured  
154 software solutions to its customers. Luckily, some already trained models were  
155 made available online and can be used to make predictions, although they are  
156 not directly contextualized for UP  $\mathcal{A}$ 's application. Because fitting customer  
157 needs must be fast, UP  $\mathcal{A}$  will directly reuse one of the prediction models as-is;  
158 *i.e.*, there will be no adaptation made whatsoever. Consequently, the computed  
159 performance predictions can be of poor quality, something that remains accept-  
160 able for UP  $\mathcal{A}$ . Typical examples of UP  $\mathcal{A}$  include start-up company engineers  
161 or fast prototyping developers, and R&D software developers that can reuse  
162 pre-trained models and want to quickly reason over performance prediction of  
163 their configurable systems.

164 In contrast, UP  $\mathcal{B}$  is not in a rush and has high expectations regarding  
165 customer satisfaction.  $\mathcal{B}$  typically wants to retrieve the best predictions so  
166 that she/he can recommend configurations fitting the users' needs. To do so,  $\mathcal{B}$   
167 creates a prediction model on-demand specifically tailored to the provided input.  
168 Typically UP  $\mathcal{B}$  corresponds to software engineers from large companies, which  
169 face high expectations in software capitalization and quality of service and can  
170 absorb the cost of building a performance model per input and workload.

171 Ultimately, UP  $\mathcal{C}$  is committed to high standards, but  $\mathcal{C}$  wants to quickly  
172 deliver high-quality software configurations. To do so,  $\mathcal{C}$  wants to find a trade-  
173 off between the two previously-identified personas.  $\mathcal{C}$  is likely to spend effort  
174 finding a pre-trained performance prediction model (or training it by using  
175 multiple inputs) so that the model can adapt to various cases. Note that in  
176 this case, even though most of the training cost is already high,  $\mathcal{C}$  wants to  
177 tailor the model to customers' specific inputs to provide high-quality predictions.  
178 Examples of UP  $\mathcal{C}$  include organizations or individuals capable of adapting  
179 pre-trained models based on the additional measurement of configurations over  
180 specific inputs.

## 181 2.2. Prediction Strategies

182 Based on these different UPs, we can distinguish several strategies for train-  
183 ing and obtaining a performance prediction model.

184 **Offline learning (rely on a pre-trained model.)** UP  $\mathcal{A}$  relies on a  
185 previously trained model, that is, a model which provides predictions before  
186  $\mathcal{A}$  comes up with an input sample. In this case,  $\mathcal{A}$  is only using a pre-trained  
187 model and does neither fine-tune the model nor retrain it nor else apply any  
188 kind of transfer learning. The model is trained on a combination of multiple  
189 software configurations and multiple input samples. We follow an input-aware  
190 setup similar to the technique proposed in [11] for a compiler of the PetaBricks  
191 language. In practice (and it is the main advantage of the method), UP  $\mathcal{A}$  does  
192 not need to compute new configurations' measurements.  $\mathcal{A}$  simply passes an  
193 input sample to the trained model that provides a prediction by computing and  
194 leveraging input properties (see hereafter, in Section section 3.1.2). Eventually,

195  $\mathcal{A}$  sends the predictions to the customers. This “pre-trained model” setting ad-  
196 vantageously reduces the computational load over UP  $\mathcal{A}$  that does not need to  
197 measure configurations. To exploit the model, only the input’s properties need  
198 to be computed and concatenated to the configurations’ descriptions used for  
199 training. (Inputs’ properties are specific to an application domain and many  
200 examples are given in the next section). The model can then be queried and  
201 all the retrieved predictions can be exploited directly. UP  $\mathcal{A}$  does not control  
202 the training process and the quality of the training set (consisting of a selec-  
203 tion of software configurations and inputs) meaning that the selections may be  
204 poorly distributed over the input space. Similarly, while querying the model,  
205 the provided input may be out of distribution, which can possibly result in weak  
206 predictions and choices.

207 **Supervised online learning (train a model on demand.)** Related to  
208 UP  $\mathcal{B}$ , the goal is to control as much as possible the quality of the prediction.  
209 In this setup,  $\mathcal{B}$  does not rely on a pre-trained model (there is no cost in the  
210 offline setting) but rather builds a performance prediction model on-demand,  
211 in an online setting. For that,  $\mathcal{B}$  has a pool of preselected software configu-  
212 rations ready to be executed with the desired inputs (*i.e.*, those coming from  
213 the customers). Another option is to have access to a software configuration  
214 sampling procedure. As soon as the input comes up, the first step consists in  
215 evaluating the performance of each selected software configuration using that in-  
216 put. Then, the prediction model is built and further predictions can be queried.  
217 Note that, unlike the previous strategy, only the desired input sample matters  
218 here, and thus there is no need to discriminate among inputs and compute in-  
219 put properties. The learned model is thus specific to the provided input, yet,  
220 predictions are expected to be more accurate than those from UP  $\mathcal{A}$  (*i.e.*, as  
221 the input diversity dimension of the problem disappeared). Yet, it supposes  
222 that UP  $\mathcal{B}$  has enough resources available to perform both the measurements  
223 and the training of the system before being able to provide predictions. Thus,  
224 in an online setting where predictions must come almost instantaneously, this  
225 strategy is probably inadequate. Also, the model is trained on-demand and for  
226 a specific input only. If multiple requests from different customers arrive at the  
227 same time, then the model will be re-trained again and again. This prevents  
228 sharing knowledge from different models and prevents capitalizing on previous  
229 training. Traditional statistical learning techniques (*e.g.*, [21, 60, 52, 3]) can be  
230 used. It boils down to address a regression problem each time a new workload  
231 or input is considered.

232 **Transfer learning (adapt pre-trained model).** Finally, UP  $\mathcal{C}$  wants to  
233 answer best to his customers but cannot afford on-demand full training predi-  
234 ction models. A suitable strategy would be to leverage the bigger cost that  
235 can be left to organizations that can provide general prediction models, and  
236 adapt, on the fly, a model to the specific input that is provided. This way,  
237 the out-of-distribution and quality of the selected software configurations and  
238 inputs for training problems would be mitigated. The adaptation would then  
239 require the general model to be retrieved so that parameters can be modified.  
240 The idea is not to retrain completely though. Two different methods can be



Approach	Description of the approach	Offline cost (Offshore Organization)	Online measurement cost (User)	Input properties	User Persona
Supervised online learning	Train performance model on demand, from scratch, each time a new input is fed to the configurable system	None	High	No	$\mathcal{A}$
Offline learning	Use a pre-trained model over measurements of multiple configurations and inputs. Input properties are used to make the prediction (in an online setting).	High	None	Yes	$\mathcal{B}$
Transfer learning	Adapt a pre-trained model for a new targeted input. It requires to gather fresh measurements of some configurations over the input (in an online setting).	Medium	Medium	Yes	$\mathcal{C}$

Table 1: Comparison of the different approaches to learn input-aware performance models of configurable systems.

241 used to adapt models, the first one is fine-tuning, such that the model weights  
242 can be adjusted quickly to the incoming input; the second one is “transfer learn-  
243 ing” [52, 26, 4, 37, 68] to find a transformation between the data distribution  
244 the model was trained for, and the data provided by  $\mathcal{C}$ ’s customers. Transfer  
245 learning can be applied for several use cases without necessarily making changes  
246 to the original model, thereby providing additional flexibility.

247 In the context of this paper, we focus on transfer learning as it addresses  
248 the out-of-distribution problem. Some transfer learning techniques have not  
249 been designed to operate over actual inputs’ changes, but rather changes of  
250 computing environments [25, 68, 26, 43, 67, 33, 24] (*e.g.*, hardware or versions).  
251 Hence, we had to design transfer learning techniques capable of leveraging the  
252 specifics of inputs. The idea is to compute a transformation allowing the transfer  
253 of the prediction capabilities from the inputs used for training to the one that  
254 comes from  $\mathcal{C}$  and vice-versa. For that,  $\mathcal{C}$  relies on a pre-trained model as  
255 does UP  $\mathcal{A}$ . Yet, as the input comes for the customer, the input’s properties  
256 are sent to the model for prediction and in the meantime,  $\mathcal{C}$  actually measures  
257 the performances of the configurations that are used for training on the new  
258 incoming input. Measured performances and predictions can then be compared  
259 to retrieve a mathematical transformation that can be applied to the pre-trained  
260 model to minimize prediction error. As said previously, the main advantage is to  
261 start from a rather general prediction model and simply adapt it. Yet it requires  
262 retrieving the model along with the software configurations that were used for  
263 training. In the end, the effort is split among online and offline settings, and  
264 between (1) the creation of the original, general model and (2) UP  $\mathcal{C}$  that has to  
265 compute the adaptation. Table 1 sums up how efforts are split between online  
266 and offline settings, UPs, and model providers for the three different prediction  
267 strategies (offline learning, supervised online learning, and transfer learning).

### 268 2.3. Research Questions (RQs)

269 Accounting for the variety of UPs and prediction strategies, we spell out the  
270 following three research questions (RQs):

271 **RQ<sub>1</sub>. How do different machine learning algorithms compare for**  
272 **establishing a relevant performance prediction model?** The production

273 of a relevant predictive performance model involves the selection of the most  
274 appropriate algorithm for that task. To address this question, we quantify the  
275 errors and the benefits of tuning hyperparameters of several relevant algorithms  
276 used in the literature. We also compare these results with a non-learning ap-  
277 proach to the problem, used as a baseline for comparison.

278 **RQ<sub>2</sub>. How to select an appropriate set of inputs for training a**  
279 **performance prediction model?** Prior to the prediction, we have to select  
280 a list of inputs whose measurements will form the training dataset. We call  
281 this process *input selection*. *RQ<sub>2</sub>* investigates what choice of input selection  
282 technique (e.g., all inputs, random selection of inputs, most diverse inputs)  
283 leads to the best results in terms of performance prediction. Depending on the  
284 offline budget, we propose and compare various input selection techniques.

285 **RQ<sub>3</sub>. How does the number of measured configurations affect the**  
286 **performance prediction models?** Since inputs and configurations interact  
287 with each other to change software performance, input selection is sensitive to  
288 the sampling of configurations *i.e.*, in the way we select the configurations used  
289 to train the model. To answer *RQ<sub>3</sub>*, we train performance models fed with  
290 different numbers of inputs and configurations.

### 291 3. Experimental protocol

292 To respond to these three research questions, we designed an experimen-  
293 tal protocol based on the following data (Section 3.1) and prediction models  
294 (Section 3.2). This section also provides details about RQ1 (Section 3.3), RQ2  
295 (Section 3.4) and RQ3 (Section 3.5).

#### 296 3.1. Data

##### 297 3.1.1. Dataset

298 We reused the measurements from 8 configurable systems and their inputs,  
299 as they are introduced in [34] and referenced in the companion repository<sup>3</sup>.  
300 Different elements are shown in Table 2. The total number of measurements  
301 taken for a software system is equal to the number of configurations multiplied  
302 by the number of inputs and the number of performance properties. For in-  
303 stance, 201 configurations of *x264* have been systematically measured along five  
304 performance properties and using 1397 videos coming from the YouTube User  
305 **General Content** (YUGC) dataset [71], for a total of 1,403,985 measures for  
306 *x264*.

##### 307 3.1.2. Using input properties to discriminate inputs

308 To differentiate the inputs directly in the learning process, we computed  
309 and added input properties [11] that describe specific characteristics of input  
310 data. The input properties are preprocessed into an input feature vector and

---

<sup>3</sup>Our data and measurement protocol are available and open

Table 2: An overview of the considered systems in their number of configurations (#Configs), configuration options (|Options|), inputs (#Input), and input properties (|Input|). It should be noted that some options have numerical values. The last column states the performance properties that were measured.

System	#Configs	Options	#Inputs	Input	Performance
<i>gcc</i>	80	5	30	7	size, ctime, exec
<i>imagemagick</i>	100	5	1000	5	size, time
<i>lingeling</i>	100	10	351	4	#conf, #reduc
<i>nodeJS</i>	50	6	1939	6	#operations/s
<i>poppler</i>	16	5	1480	6	size, time
<i>SQLite</i>	50	3	150	8	15 query times q1-q15
<i>x264</i>	201	23	1397	7	size, time, cpu, fps, kbs
<i>xz</i>	30	4	48	2	size, time

311 concatenated with the configuration features. Input and configuration options  
 312 jointly form the feature vector that is passed as the input to the machine learning  
 313 model.

314 The encoding into a single vector allows statistical learning techniques to  
 315 operate over a unified encoding to predict performance. Though both are en-  
 316 coded as features, configuration options and input characteristics refer to and  
 317 describe different entities (*i.e.*, inputs and software configurations respectively).  
 318 Configuration options directly come from the software and the development ac-  
 319 tivity. They are used to differentiate every single configuration that can be  
 320 built. Options have been made explicit, traced back and implemented in the  
 321 code by developers and/or domain experts. On the other hand, characteristics  
 322 describing inputs are not necessarily made explicit and usually require domain  
 323 experts to model features that should be both descriptive about the content,  
 324 helpful for discriminating one input from the other, and also informative for  
 325 predicting performance. They might not be as differentiating as the ones for  
 326 configurations as different inputs may result in the same characteristics. In the  
 327 end, paired with the configuration options, we expect that this feature vector  
 328 allow us to observe very similar performances from the systems. Some learning  
 329 approaches can leverage these input properties as predictors (or features) when  
 330 building or adapting prediction models. Based on domain knowledge, we list  
 331 hereafter what input properties have been computed for the different sorts of  
 332 inputs (see also Table 2): for the *.c* scripts compiled by *gcc*, the size of the file,  
 333 the number of imports, methods, literals, *for* and *if* loops and the number of  
 334 lines of code (LOCs); for the images fed to *imagemagick*, the image size, width  
 335 and height, category (describing the image content, e.g., ostrich, dragonfly, or  
 336 koala), and its averaged (*r*, *g*, *b*) pixel value; for SAT formulae processed by  
 337 *lingeling*, the size of the *.cnf* file, the number of variables, *or* operators, and  
 338 *and* operators; for the test suite of *nodejs*, the size of the *.js* script, the LOCs,  
 339 number of functions, variables, *if* conditions, and *for* loops; for the *.pdf* files  
 340 processed by *poppler*, the page height and width, the image and pdf sizes, the

341 number of pages and images per input pdf; for databases queried by *SQLite*, the  
 342 number of lines for eight different tables of the database; for input videos en-  
 343 coded by *x264*, the spatial, temporal, and chunk complexity, the resolution, the  
 344 encoded frames per second, the CPU usage, the width and height of videos <sup>4</sup>;  
 345 for the system files compressed by *xz*, the format and the size.

### 3.1.3. Separation Training-Test

346 We randomly split each set of configurations into a training set and a test  
 347 set. We repeat this with varying proportions of configurations in the training  
 348 set – we start with 10% of configurations dedicated to training and then 20%,  
 349 30%, . . . , up to 90%. The training set is available for data selection and training  
 350 of the models, whereas the test set is purely dedicated to evaluating the trained  
 351 models. To avoid biasing the results with different samplings of configurations,  
 352 we fix the random seeds so the different techniques work with the same training  
 353 and test sets. Note that every training is done independently from one model  
 354 to another and from scratch so that all training procedures do not share any  
 355 information and start from the same point.  
 356

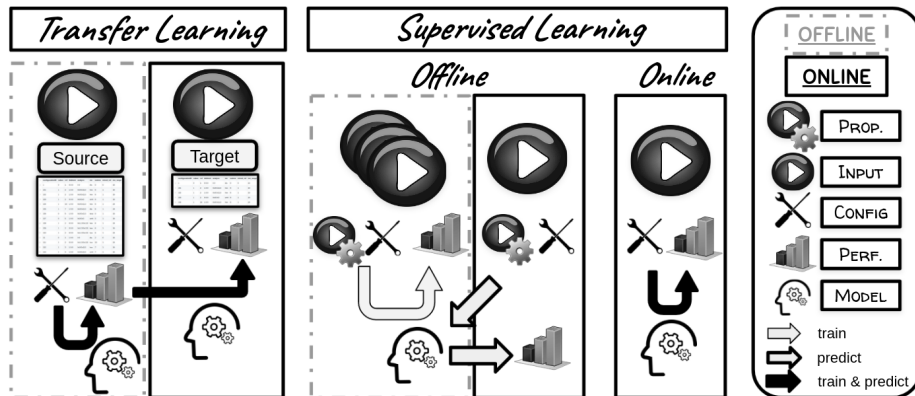


Figure 2: Learning approaches to address the performance prediction problem

## 3.2. Performance Prediction Models

357 In this section, we define the different predictive performance models used  
 358 in the rest of the paper.  
 359

### 3.2.1. Strategies and baselines

360 In addition to the three learning strategies that we described in Section 2.2  
 361 (supervised online learning, offline learning, transfer learning), we consider a  
 362 baseline that does not learn the specifics of the inputs. The approach, called  
 363

<sup>4</sup>For *x264*, input properties were already computed in [71]

364 **Average**, simply computes the average of configurations' measurements ob-  
365 served on the input. This is supposed to mimic the behaviour of an end-user  
366 that might consider first a configuration of a system (e.g., the default configu-  
367 ration [76]) and then slightly explore the configuration space to have an average  
368 of the performance values. This way, the end-user may measure various configu-  
369 rations and approximate the trend of performance values. Average is a better  
370 approximation than the systematic use of a single point (e.g., default value)  
371 when it comes to predicting the performance of any configuration, especially  
372 when inputs change.

373 An advantage of this approach is that there is no cost for an offshore organi-  
374 zation (i.e., the ones that provide pre-trained models). Instead, the baseline  
375 applies on demand for every new input (in an online setting). Once the input  
376 is received, a set of randomly sampled configurations is executed over the in-  
377 put to retrieve performance measures. The average of these performances is  
378 returned, and these are the values that are communicated to the users in the  
379 sense that they can expect such performance on average. Obviously, such an  
380 averaged configuration does not account for possible variations in the perfor-  
381 mance of configurable systems. In particular, the average value remains fixed  
382 regardless of the configuration we wish to predict performance for, which could  
383 possibly lead to a high degree of inaccuracy.

### 384 3.2.2. Learning Algorithm

385 Until now, we described the learning strategies but did not mention which  
386 learning algorithms we were using. We chose 4 different algorithms, namely:

- 387 1. OLS Regression [57] from Scikit-learn [51], estimating the performance value  
388 with a weighted sum of variables. It is not designed for complex cases;
- 389 2. Decision Tree [55] from Scikit-learn, using decision rules to separate the con-  
390 figurations into sets and then predicting the performance separately for each  
391 set;
- 392 3. Random Forest [49] from Scikit-learn, an ensemble algorithm based on bag-  
393 ging, combining the knowledge of different decision trees to make its predic-  
394 tion;
- 395 4. Gradient Boosting Tree [18] from XGBoost [10], also derived from Decision  
396 Tree. Unlike Random Forest training different trees, gradient boosting aims  
397 at improving one tree by specialising its rules of decision at each step.

398 These are prevalent algorithms and are commonly used for tabular data. We  
399 do not include deep learning or neural networks since we do not have lots of  
400 measurements in an online setting, which is usually a requirement, and since  
401 these methods are still commonly outperformed by random forests or gradient  
402 boosting [58, 20]. We train each machine learning algorithm with its default  
403 parameters, if not stated otherwise. The optimal parameters for an algorithm  
404 are dependent on the dataset, its size, and the performance property. As a  
405 result, they are necessary to be adjusted on a per-case basis.

406 *3.3. Selecting Algorithms (RQ<sub>1</sub>)*

407 First, we address RQ<sub>1</sub> - *How do different machine learning algorithms com-*  
408 *pare for establishing a relevant performance prediction model?* We decompose  
409 it into three parts, that jointly answer the research question.

410 *3.3.1. Why using machine learning?*

411 Our first goal is to state whether machine learning is suited to address the  
412 performance prediction problem. To assess the benefit of using machine learn-  
413 ing, we compare the *Average* baseline to different learning algorithms. We im-  
414 plement them in an online setting, *i.e.*, we use the supervised online approach;  
415 given the input of the user, we want to estimate its performance distribution.  
416 This is a prediction for one input at a time.

417 *3.3.2. Which machine learning algorithm to use?*

418 This evaluation is also the opportunity to compare these learning algorithms  
419 and search for the one that outperforms the others. We also study the evolu-  
420 tion of their prediction errors with increasing training sizes. We consider those  
421 listed in Section 3.2.2. After training them on the training set, we predict the  
422 performance distribution of the test set and compute the prediction error. We  
423 repeat it for all combinations of system, inputs, and performance properties.  
424 As prediction error, we rely on the Mean Absolute Percentage Error [40]. In  
425 Figure 3, we display the average MAPE values for various training sizes.

426 *3.3.3. What is the benefit of hyperparameter tuning?*

427 Finally, we want to estimate how much accuracy we could expect to gain  
428 when we tune the hyperparameters of learning algorithms. To do so, we rely  
429 on a grid search [5] for hyperparameter tuning. We compute the training du-  
430 ration and prediction errors of these algorithms, with and without tuning their  
431 hyperparameters, and report the average difference for both.

432 *3.4. Selecting Inputs (RQ<sub>2</sub>)*

433 RQ<sub>1</sub> studies the effectiveness of machine learning. However, different ways  
434 of selecting inputs during the data collection or a different number of inputs  
435 could alter the accuracy of the final performance model. Then, we address RQ<sub>2</sub>  
436 - *How to select an appropriate set of inputs for training a performance prediction*  
437 *model?* We separate this into three questions.

438 *3.4.1. How many inputs do we need to learn an accurate predictive performance*  
439 *model?*

440 From the perspective of a user in charge of the training, adding an input to  
441 measure incurs a computational cost and should be justified by an improvement  
442 of the model. So, what is the effect of adding new inputs on the accuracy of  
443 predictive performance models? How many inputs do we need to reach a decent  
444 level of accuracy? We aim at minimising the number of inputs used in the  
445 training while maximising the accuracy of the obtained model. To do so, in

446 this part of the evaluation, we train different performance models using various  
447 numbers of inputs and compare their prediction errors.

448 Then, once the number of inputs is fixed, we search if there is any benefit  
449 in precisely and methodologically selecting the inputs for data collection and  
450 model training. Does it bring any improvement over the random selection of  
451 inputs? Do the different inputs used in the training set change the final predic-  
452 tive performance model accuracy? Depending on the offline budget of the user,  
453 we have different objectives.

#### 454 3.4.2. How to select the input data for an offline setting?

455 If the offline budget is restricted, a.k.a. the *offline setting*, the goal is to  
456 constitute a representative set of inputs to learn from, in order to build a per-  
457 formance model that will generalise as much as possible. We care about selecting  
458 diverse and representative inputs, in order to predict accurate results whatever  
459 the input data. For this setting, we compare the following input selections:

- 460 1. Random - Using a uniform distribution to decide which inputs should be  
461 included in the training;
- 462 2. K-means - Based on input properties, we apply K-means clustering to dif-  
463 ferentiate clusters of inputs with distinct characteristics. To increase the  
464 diversity in the selection, we pick the inputs closest to the center of the clus-  
465 ters;
- 466 3. HDBScan - Similar to the previous technique but with another clustering  
467 algorithm, namely the HDBScan [6], using density-based instead of means-  
468 based<sup>5</sup>;
- 469 4. Submodular (Selection) - This technique computes a similarity matrix be-  
470 tween the different inputs of a software system and optimises facility location  
471 functions [31] to choose a representative set of inputs<sup>6</sup>.

472 For this *offline setting*, we implement the supervised offline approach with a  
473 Gradient Boosting (best in Section 4.1.2). Once the input selection technique  
474 chose the input, we include all related measurements in the training set. The  
475 test set is then composed of the measurements of all other inputs, not selected.  
476 To avoid biasing the machine learning model with different scales of performance  
477 distribution, we choose to standardise [13] all performance properties. But it  
478 has a drawback: since their values are close to zero, it artificially increases  
479 the MAPE values. To overcome this, we switch to the Mean Absolute Error  
480 (MAE) [40]. Since the performance property is standardised, we assume that  
481 models with MAE values inferior to 0.2 are good – way better than the expected  
482 average distance  $\frac{2}{\pi} \simeq 1.13$  [64] between two points selected uniformly. We repeat  
483 the prediction 20 times and depict the average MAE (y-axis, left) in Figure 4  
484 for different input selection techniques (lines) and number of inputs (x-axis) on  
485 a per-system basis. We added the number of training samples (y-axis, right).  
486 Except for *gcc* and *xz*, x-axis are in log scale.

---

<sup>5</sup>We rely on this implementation: <https://hdbscan.readthedocs.io/> [39]

<sup>6</sup>We rely on this implementation: <https://apricot-select.readthedocs.io/> [56]

487 *3.4.3. How to select the input data for an online setting?*

488 If the offline budget is low, a.k.a., in the *online setting*, then we must be  
489 efficient and focus on predicting the performance distribution for the current  
490 input of the user. For this setting, we implement a transfer learning approach:  
491 the input of the user becomes the target input, and the candidate input becomes  
492 the source input. In this online setting, the goal of input selection becomes to  
493 find one good source input that is as close as possible to the current input  
494 of the user - in terms of characteristics and performance. The choice of a  
495 good source should improve the performance prediction of the transfer learning  
496 approach [32]. We propose the following input selections:

- 497 1. Random - Same baseline as in the offline setting;
- 498 2. Closest (Input) Properties - Two inputs sharing common characteristics might  
499 also share common performance distributions. Following this reasoning, to  
500 improve the performance prediction, we have to select an input whose prop-  
501 erties are similar to the current input’s properties. To do so, we compute the  
502 MAE between the properties of the current input and the properties of all  
503 the candidate inputs. We pick the input obtaining the smallest MAE value;
- 504 3. Closest Performance - We use the few measurements already measured on the  
505 current input. For these, we compute the Spearman correlation [27] between  
506 the performance distribution of the current input and all the candidate inputs.  
507 Finally, we select the candidate with the highest correlation;
- 508 4. Input Clustering (& Random) - With the help of a K-Means algorithm, we  
509 form different clusters of inputs based on their properties. We randomly pick  
510 a candidate input in the cluster of the current input.

511 For this question, we use Gradient Boosting. We display the median MAPE  
512 results over 10 predictions for all software systems, performance properties and  
513 number of inputs in Table 3.

514 *3.5. Selecting Configurations (RQ<sub>3</sub>)*

515 In this section, we vary the budgets of (1) inputs and (2) configurations  
516 used to constitute the training set fed to the model. *RQ<sub>3</sub> - How does the*  
517 *number of measured configurations affect the performance prediction models?* In  
518 this research question, we compare the accuracy of models according to the  
519 numbers of inputs and configurations used during their training, answering these  
520 questions:

521 *3.5.1. What is the best tradeoff between selecting inputs and sampling configu-*  
522 *rations?*

523 For a fixed number of configurations, we study the evolution of the accuracy  
524 with the number of considered inputs. Is it better to measure lots of config-  
525 urations or numerous inputs? Since the evaluation is designed to improve the  
526 generalization of the model, it mostly relates to models trained in an offline  
527 setting. Therefore, we implement the offline approach, fix the algorithm to  
528 Gradient Boosting and use the random baseline as input selection technique.  
529 We repeat the experiment 20 times. In Figure 5, we depict the MAE (colour)  
530 for various numbers of inputs (x-axis) and configurations (y-axis).



531 We are then interested to look at the impact of the number of configurations  
532 on the accuracy of three input-aware approaches (transfer learning vs supervised  
533 online; offline learning vs supervised online).

534 *3.5.2. Is it better to use the transfer learning or the supervised online approach?*

535 Depending on the online budget of the user, it can be worth (or not) to  
536 transfer the knowledge from one input to another. If the online budget is high,  
537 we guess there is no need to transfer, *i.e.*, we do not use transfer learning. If  
538 this budget is low, we can benefit from the transfer learning approach. How  
539 much online budget justifies the decision for transfer learning? To answer this,  
540 we implement both approaches with different numbers of configurations on the  
541 target input. We use Gradient Boosting and predict the performance spanning  
542 all systems and performance properties. Due to outliers drastically increasing  
543 the average value, we compute the median MAPE instead of the average. In  
544 Figure 6, we display the MAPE value (y-axis) for different budgets of configu-  
545 rations (x-axis).

546 *3.5.3. Should we train performance models offline or online?*

547 To answer this question, we compare the supervised offline approach to the  
548 supervised online approach w.r.t. MAE. We predict performance implementing  
549 both approaches and compute the MAE value for different training size – varying  
550 from 10% to 90% of the configurations available per input. Figure 7 displays  
551 the results for both approaches, *i.e.*, the average MAE on all software systems  
552 and performance properties.

## 553 4. Evaluation

554 We report the results following the protocol of Section 3.

555 *4.1. Selecting Algorithms (RQ<sub>1</sub>)*

556 *4.1.1. Is using ML relevant in the context of predictive performance modelling?*

557 Figure 3 shows the benefits of using ML as compared to the *Average* base-  
558 line. It appears that ML techniques clearly outperform the average performance  
559 value predicted by the baseline for all training sizes. The key indicator to study  
560 is the evolution of errors with increasing training proportions; while the base-  
561 line’s accuracy does not progress with additional measurements, the learning  
562 algorithms improve their prediction, from 12% to 3% error.

563 *4.1.2. Which ML algorithm to use?*

564 While ML is generally beneficial, the prediction quality varies between the  
565 different algorithms. For instance, OLS Regression generally leads to bad re-  
566 sults, *e.g.*, 22% of error with 10% of the configurations. Unlike the OLS re-  
567 gression, tree-based learning algorithms take advantage of the addition of new  
568 measurements. For a budget of 50% of the configurations, their median pre-  
569 diction goes under the 5% of error, which is encouraging. This result of 5% is

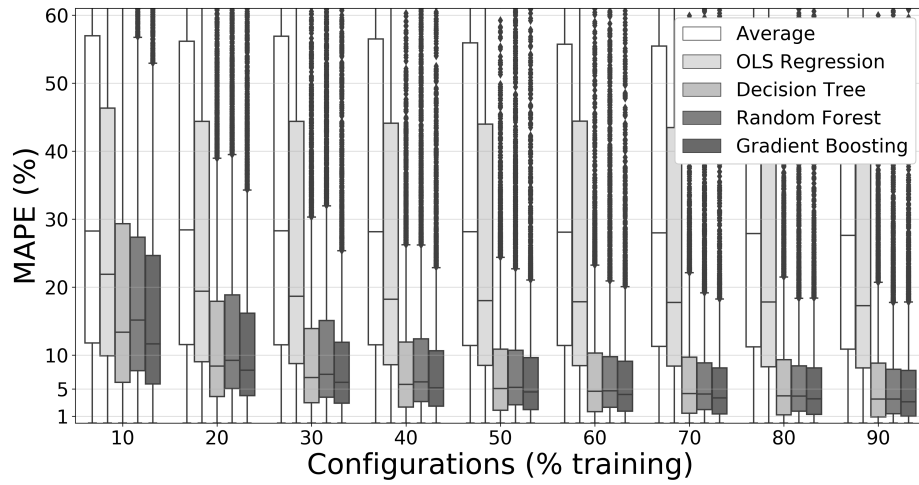


Figure 3: Which (learning) algorithm to use?

570 only valid in average; the prediction will be better for a few software systems,  
 571 *e.g.*, *imagemagick* or *x264*, but does not hold for others, *e.g.*, *lingeling* or *pop-*  
 572 *pler*. Though there is no big difference between these three learning algorithms,  
 573 we observe slightly better predictions for Random Forest compared to Decision  
 574 Trees, and for Gradient Boosting compared to Random Forest.

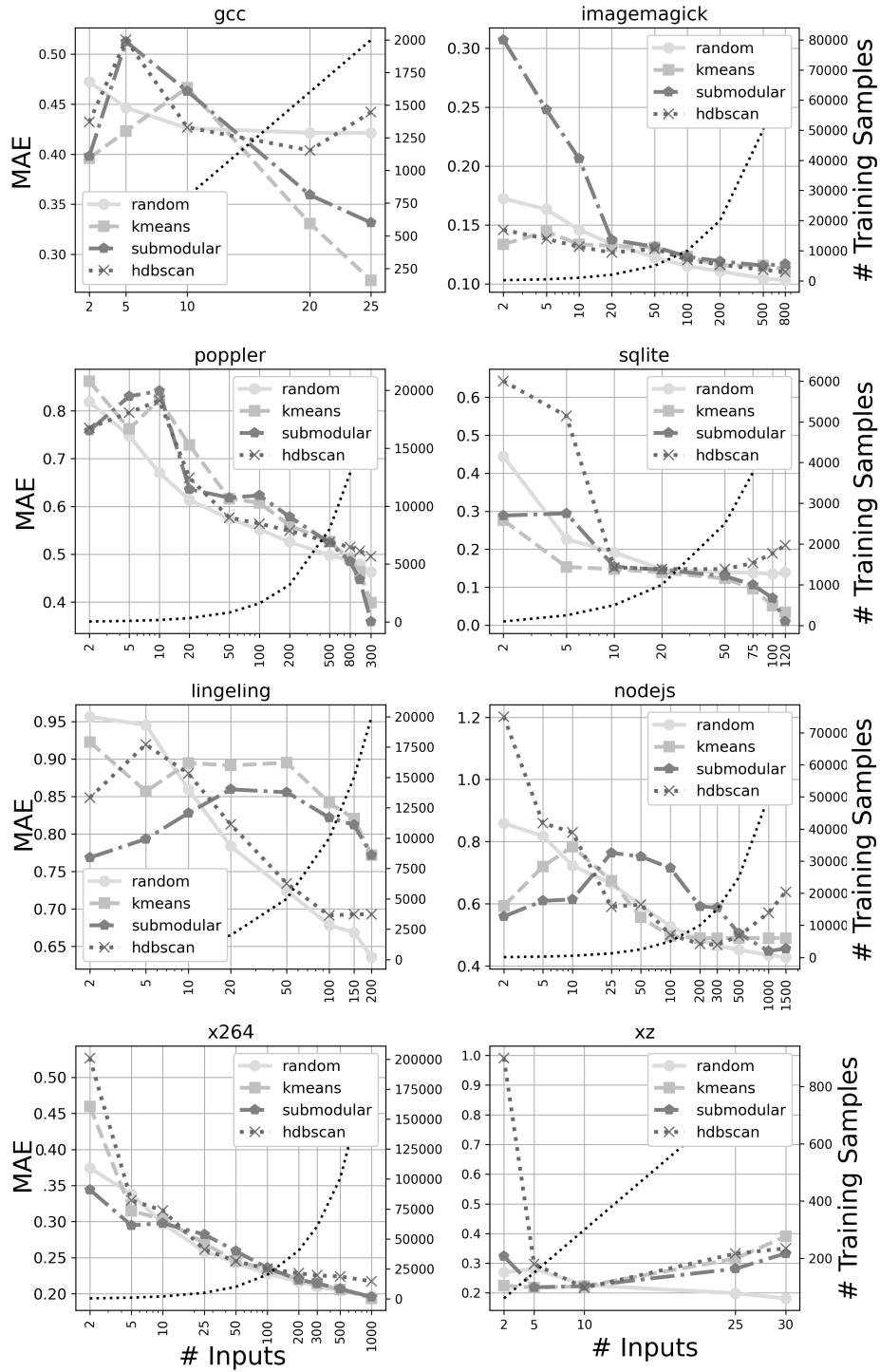


Figure 4: Offline setting with Gradient Boosting (best) and different input selection techniques: Influence of input selection and the number of inputs (lower MAPE = better).

575 *4.1.3. What is the benefit of hyperparameter tuning?*

576 Our results found that hyperparameter search improves the MAPE on av-  
577 erage by  $8 \pm 16\%$  within a range of  $3 \rightarrow 37\%$  but also requires on average 120  
578 times more training time when doing a grid search of estimator parameters. It  
579 should be noted that this is an improvement in percent, not percentage points.  
580 While the exact overhead of hyperparameter search is dependent on the num-  
581 ber of configuration options of the model and the search method, we note that  
582 the overall benefit on the datasets is limited. This is especially confirmed by  
583 the observation that the best-found hyperparameters were changing depending  
584 on both the system and the size of the training dataset. For simplicity of the  
585 setup, the rest of the evaluation uses the default parameters of each model, if  
586 not otherwise noted.

587 **RQ<sub>1</sub>** Machine learning is well-suited to address the predictive perfor-  
mance modelling problem over configurations and inputs. It outper-  
forms the *Average* baseline as it makes more accurate predictions. Our  
results also show that using tree-based learning should be favoured over  
OLS Regression. These tree-based algorithms reach decent levels of errors  
with reasonable online budgets, between 5% and 10% relative error  
for most of the cases, and potentially improved by 8% when tuning hy-  
perparameters.

588 *4.2. Selecting Inputs (RQ<sub>2</sub>)*

589 *4.2.1. How many inputs are needed to learn an accurate predictive performance*  
590 *model?*

591 Measuring inputs differs across software systems: measuring 5 inputs rep-  
592 represents  $5 * 201 = 1005$  configurations for *x264* but only  $5 * 30 = 150$  for *xz*.  
593 Figure 4 shows that the performance model reaches its lowest error threshold  
594 when considering about 20 inputs. Results are thus easier to interpret on sys-  
595 tems with more inputs, as compared to *gcc* and *xz*. There are however more  
596 difficult cases, *e.g.*, *Node.js* and *poppler*, in our experiment that might require  
597 more inputs to improve the accuracy of the prediction. To get a consistent pre-  
598 diction, we recommend the user to measure at least 25 inputs with a sufficient  
599 number of configurations per input.

600 *4.2.2. How to select the input data for an offline setting?*

601 In an offline setting, we seek to train a generalized model for all inputs; the  
602 selected inputs are supposed to be representative of the diverse set of inputs to  
603 be expected during deployment. Figure 4 presents our results. As expected,  
604 with an increased number of selected inputs, the influence of the input selection  
605 decreases. The input selection is especially important for small numbers of  
606 inputs. The evaluation shows that our techniques kmeans, submodular and  
607 hdbscan fail to beat the random baseline when selecting the inputs prior to

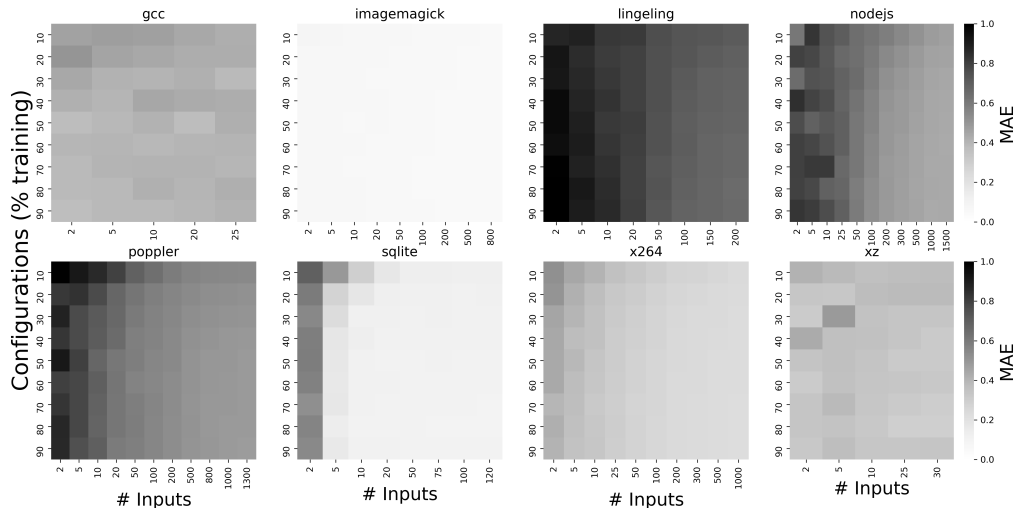


Figure 5: Offline setting with Gradient Boosting (best) and random baseline as input selection technique: Error of performance models (lower MAE = brighter color = better) according to different budgets (inputs & configurations). For imagemagick a consistent avg. MAE of 0.06 is observed.

608 the training of the model. There is no clear outperforming technique of input  
 609 selection. These results could be explained by multiple factors: (1) the input  
 610 properties processed by the input selections are not sufficient to differentiate  
 611 the inputs (2) our baseline focuses on selecting different profiles of inputs, while  
 612 it may be more efficient to select a set of average-like inputs. Out of this result,  
 613 we advise keeping it simple and adopting the random baseline.

#### 614 4.2.3. How to select the input data for an online setting?

615 In an online setting, we specifically build a model for the current input and  
 616 select a similar input to transfer the knowledge from. Table 3 details the re-  
 617 sults for the different input selections, as the median MAPE results over 10  
 618 predictions for all software systems, performance properties, and the number of  
 619 inputs. Unlike the offline setting, our input selection techniques were able to  
 620 beat the random baseline, the best input selection technique being the Closest  
 621 Performance with an average MAPE around 3.8, followed by the Closest Proper-  
 622 ties (4.2) and the Input Clustering (4.7). Wilcoxon signed-rank tests [59] (with  
 623 significance levels at 0.05) confirm that predictions related to different input se-  
 624 lections are significantly different from those using the random baseline:  $p = 0.0$   
 625 for Closest Performance,  $p = 1 * 10^{-184}$  for Closest Properties and  $p = 1 * 10^{-27}$   
 626 for the Input Clustering. Therefore, and to continue to provide guidance for  
 627 users, we advise using the Closest Performance to select the input in an online  
 628 setting. But beyond the raw comparison of error values, beating the Random  
 629 baseline with the Closest Property technique is a strong result. Empirically,  
 630 it validates that these input properties are valuable to compute and should be

Table 3: Online Setting - Influence of input selection

Input Selection	MAPE (%)	Training Time (sec)
Random	5.22	0.02
Closest Properties	4.17	0.05
Closest Performance	3.82	0.07
Input Clustering	4.70	0.02

631 included in the models to improve the prediction. Besides, the evaluation shows  
632 that training times are negligible.

633 **RQ<sub>2</sub>** In an offline setting, the results show that diversification of inputs (rather than configurations) should be prioritized by using uniform distribution to select inputs, *i.e.*, the random baseline. In an online setting, the performance correlations technique gains about 1.4 point of error compared to a random selection of inputs. Our results empirically validate the important role of input properties when predicting software performance, *i.e.*, pretrained performance models (offline) can be reused under the condition input properties are computed and leveraged on-line.

### 634 4.3. Selecting Configurations (RQ<sub>3</sub>)

#### 635 4.3.1. What is the best trade-off between selecting inputs and sampling configurations? 636

637 According to Figure 5 results, diversifying the inputs is more effective than  
638 selecting different configurations to train an input-aware performance model.  
639 But for a fixed budget of inputs, there is a slight improvement in accuracy when  
640 increasing the number of configurations. As a result, both should be combined  
641 to obtain the best possible model. Overall, the ideal budget – both in terms  
642 of inputs and configurations – highly depends on the expected level of errors  
643 combined with the difficulty of learning a predictive performance model for the  
644 software under test. For instance, predicting the performance of *imagemagick*  
645 is relatively easy, with an average MAE value at 0.06. For this software system,  
646 picking 25 inputs is almost already a waste of resources, we do not need that  
647 much data - 5 inputs is already enough with 30% of the configurations. Other  
648 systems are harder to learn from *e.g.*, *lingeling* with an average MAE of 0.76.  
649 For these, we recommend increasing the number of inputs and configurations.  
650 The MAE obtained on the training set should be used as a proxy to estimate the  
651 difficulty of predicting the performance of the software under test. The greater  
652 its value, the greater the budget needed to learn an accurate model.

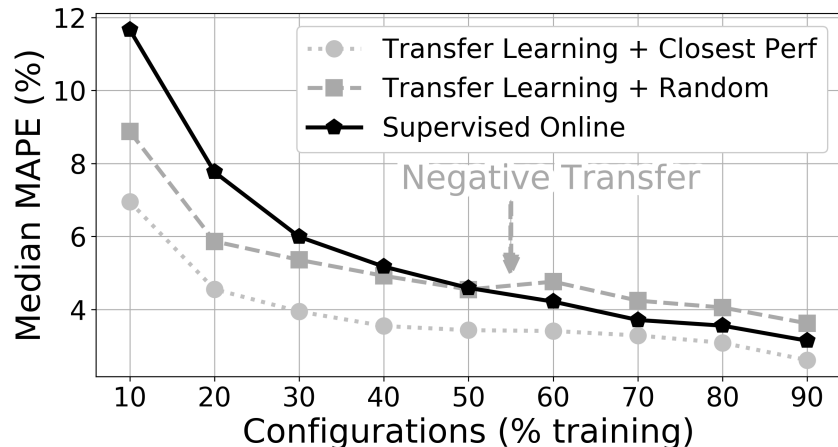


Figure 6: Transfer Learning VS Supervised Online Learning.

653 *4.3.2. Is it better to use transfer learning or a supervised online approach?*

654 With the input selection technique set to Closest Performance and whatever  
 655 the percentage of configurations used in the training, the transfer approach al-  
 656 ways outperforms the supervised online approach, as shown in Figure 6. This  
 657 strong result demonstrates the importance of capitalizing on the existing mea-  
 658 surements, measured in an offline setting. Hence, if we are able to create rep-  
 659 resentative sets of inputs for each software system (thanks to Closest Perf-  
 660 mance), then transfer learning becomes the best approach to use.

661 However, the strategy of picking the source input among all the inputs of our  
 662 dataset is not always possible – it requires a high offline budget. We consider  
 663 another scenario where we put ourselves in the situation of a user with a low  
 664 offline budget *i.e.*, not able to select an ideal source input. We thus add the  
 665 comparison of transfer learning with a random input selection baseline. Even in  
 666 this case, we still outperform the supervised approach for less than 55% of the  
 667 configurations. But this transfer with random selection has an expiration date;  
 668 after a training proportion of 55% - represented by the arrow on the graph,  
 669 it leads to negative transfer: the added measurements (of the source) become  
 670 noisy data interfering with the training of the target model.

671 *4.3.3. Should we train performance models offline or online?*

672 Figure 7 shows the evolution of the two supervised approaches, offline and  
 673 online, depending on how many configurations are used as part of the train-  
 674 ing. The first point we notice is the slow progression of the supervised offline  
 675 approach, only from 0.37 to 0.30 between 10% and 90% of configurations.

676 A possible explanation is that it is so hard to generalize over the input dimen-  
 677 sion that it hides the benefit of adding configurations. While when considering  
 678 only one input at a time, this difference in performance distributions does not  
 679 bother the training of the machine learning model. Nevertheless, comparing the

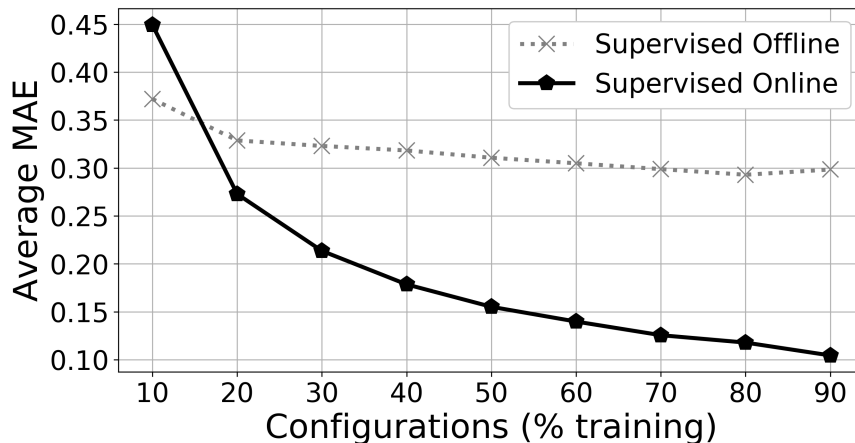


Figure 7: Offline Learning VS Supervised Online Learning.

680 raw numbers provides a straight answer to the initial question: unless the online  
 681 budget is really low, if the choice between a supervised offline and a supervised  
 682 online approach occurs, one should definitely prefer the online approach. But  
 683 this finding has to be contextualized w.r.t. cost and effectiveness. From the  
 684 point of view of the final user, the computation of measurements in an online  
 685 setting to build a performance model will always last longer than offline predic-  
 686 tion, using an already-trained model. Yet, when users have an online budget  
 687 (even a small one), they should always prefer the online approach compared to  
 688 the offline approach. Stated differently, the supervised offline approach should  
 689 be adopted for lack of a better solution, as the last approach to implement when  
 690 users cannot afford to measure configurations in an online setting.

691 ***RQ<sub>3</sub>*** Supervised online learning quickly outperforms the offline learning  
 version. With more than 20% of configurations for training, online  
 learning already shows a lower MAE on average than its offline counter-  
 part. We come up with the following high-level recommendation: 1) if  
 the online budget is low, transfer learning should be used; 2) with a sub-  
 stantial online budget (55% of the configurations in our experiments), it  
 might be better to use the supervised online approach; 3) offline learning  
 without transfer should be avoided, except for very small online config-  
 uration budgets.

## 692 5. Discussion

693 Each part of the evaluation provides a recommendation for the three user  
 694 profiles defined in Section 2.1, depending on the trade-off between their offline



695 and their online budgets. This discussion summarizes our findings while answer-  
696 ing  $RQ_1$  to  $RQ_3$  and turns these findings into recommendations and actionable  
697 rules, thus guiding the user to solve the performance prediction problem. **How**  
698 **to help users predict their software performance, whatever be the**  
699 **input data and their configuration?**

700 Depending on the available online budget, we distinguish the following cases:

- 701 • If the user has a **high online budget** (*e.g.*, user persona  $\mathcal{C}$ ) we recommend  
702 using the supervised online approach (Sec. 4.3.3) with a Gradient Boosting  
703 Tree implementation (Sec. 4.1.2) and tuned hyperparameters (Sec. 4.1.3). In  
704 that case, users can expect low prediction errors. Assuming configurations’  
705 measurement has been collected, learning a performance model from scratch  
706 for each unique input is the ideal scenario, as is the case with a large online  
707 budget.
- 708 • If the user has a **low online budget**, we can also recommend the super-  
709 vised online approach, but cannot promise outstanding performance estima-  
710 tions. Hence, if a representative set of inputs has already been measured *i.e.*,  
711 with a big offline budget (as for user persona  $\mathcal{B}$ ) we rather recommend using  
712 the transfer learning approach (Sec. 4.3.2) with a Gradient Boosting algorithm  
713 and using the closest performance input selection technique (Sec. 4.2.3). Our  
714 experiments show that the performance predictions of user persona  $\mathcal{B}$ , who is  
715 counting on inputs in an offline setting, will outperform the online predictions of  
716 user persona  $\mathcal{C}$  whatever be the budget of configurations for reasonable budgets  
717 of configurations;
- 718 • If the user has **no online budget** (*e.g.*, the user persona  $\mathcal{A}$ ), then the available  
719 offline budget is key. If the offline budget is low, there is no silver bullet:  
720 since we cannot guarantee low errors with our models, it is probably better  
721 to avoid predicting than providing a poor estimation of software performance.  
722 If the user has access to a diverse set of configurations’ measurements over  
723 different inputs (high offline budget), then we can advise using the supervised  
724 offline approach (Sec. 4.3.3) implementing a Gradient Boosting algorithm with  
725 a random selection of inputs (see Sec. 4.2.2).

726 Figure 8 summarizes these rules of thumb into a flow diagram. We also  
727 depict likely locations for user personas  $\mathcal{A}$ ,  $\mathcal{B}$  and  $\mathcal{C}$  at the end of the decision  
728 process, based on our previous recommendations, as well as the observed relative  
729 errors from our experiments. These observed errors serve as a rule-of-thumb,  
730 but are, of course, not directly transferable to other systems and setups.

731 As a limitation of our work, we highlight that it is difficult to learn the  
732 performance distribution for a few software systems *e.g.*, *lingeling*, *poppler*, and  
733 even *Node.js*. For these systems, the prediction errors are above 20% when im-  
734 plementing a supervised offline approach with tight budgets of configurations  
735 or inputs. It is worth noticing that a computational effort in terms of measure-  
736 ments is requested *i.e.*, to measure more than the general and averaged threshold  
737 of 25 inputs on average. The requested amount of inputs on a per-system basis

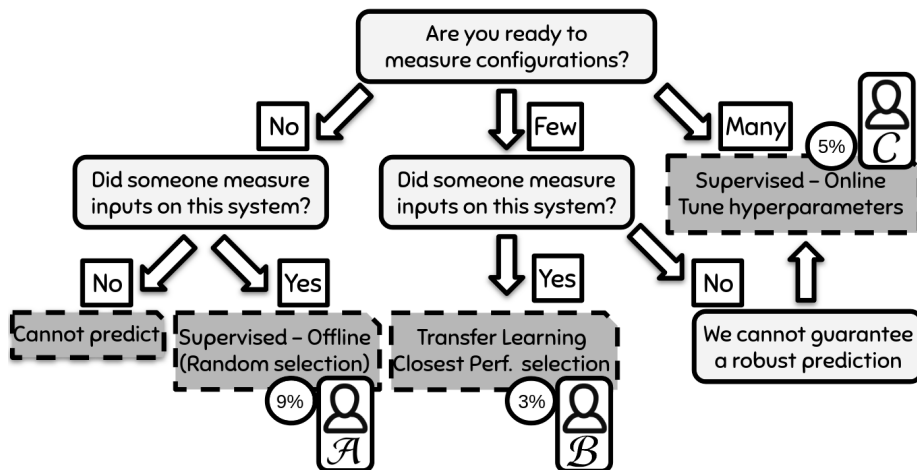


Figure 8: Lessons Learned - A flow diagram for users

738 is documented in the companion repository<sup>7</sup>. This is potentially related to the  
 739 impact of individual features on the performance metric, i.e. the spread of the  
 740 correlation, which was investigated in concurrent work for the same dataset as  
 741 used in this paper [34]. For the difficult-to-learn software systems, there are  
 742 multiple features that have a high impact on the performance metric [34, Table  
 743 4], leading to a more difficult-to-learn objective landscape for the regression  
 744 task.

745 Our results relate further to the existing body of work and confirm some  
 746 of the results previously found. For example, BEETLE [32] highlights the im-  
 747 portance of selecting the right input-specific source(s) for transfer learning to  
 748 maximize the accuracy and mitigate the risk of negative transfer, a problem  
 749 similar to the input selection we consider in  $RQ_2$ . In [26], it is discussed in  
 750 which scenarios transfer learning is more applicable compared to when it might  
 751 lead to a negative transfer. The applicability is found to be hindered by a  
 752 higher severity of the change between the original training environment and the  
 753 target prediction environment to which the model is transferred. This prob-  
 754 lem of negative transfer in variability modelling was similarly confirmed for the  
 755 Linux kernel in [37]. The negative transfer problem underlines the importance  
 756 of establishing a training set that is as broad and diverse as possible, in order  
 757 to best fit the data distribution that the model will apply during deployment.  
 758 The closer the target environment, albeit new inputs, configurations, or other  
 759 variability aspects, is to the environments from which the training data was col-  
 760 lected, the smaller the risk of a negative transfer and the better the performance  
 761 prediction quality.

<sup>7</sup>See [https://github.com/simula-vias/input-aware-performance-models/blob/main/src/when\\_to\\_stop\\_measuring\\_inputs.ipynb](https://github.com/simula-vias/input-aware-performance-models/blob/main/src/when_to_stop_measuring_inputs.ipynb)

## 762 6. Threats to Validity

763 A first threat to validity is linked to the data we are using; since we rely  
764 on a dataset [34], we are exposed to the same threats to validity. In particu-  
765 lar, an error in the measurement protocol could invalidate our results. Besides,  
766 we do not consider all the possible configuration options of the software sys-  
767 tems. The fact that it includes multiple software systems (8 in total) and a  
768 consequent number of performance measurements (roughly two million when  
769 considering the different performance properties) is supposed to alleviate these  
770 threats. A second threat to validity relates to the input properties computed in  
771 Section 3.1.2. Since we are not domain experts of each of the eight software sys-  
772 tems considered in this experiment, we cannot validate the construction of such  
773 properties, *i.e.*, it is likely that there is an opportunity to craft more expressive  
774 input properties. To the best of our knowledge, which input properties to use in  
775 order to improve the performance prediction remains an open question [11, 75].  
776 Furthermore, we neglect their computational cost. As we mentioned, being able  
777 to report precisely the properties can be a tedious problem. Measures need  
778 to be precise, external factors need to be mitigated as much as possible to re-  
779 duce potential interactions with the measurements, *etc.* Being meticulous about  
780 these aspects may drastically increase the cost to get such measurements and  
781 ultimately threaten the results of Section 4.2.2 or overestimate the benefit of the  
782 supervised offline approach. Another threat to validity is related to the random-  
783 ness in machine learning methods, subject to modifications in their predictions.  
784 To reduce these stochastic effects, we (1) fix the random seed to feed the same  
785 training and test sets to all models and have comparable results and (2) repeat  
786 the experiments 20 times. Finally, we acknowledge that we relied on already  
787 existing libraries and implementations. These can be buggy or present some in-  
788 accuracies that may favour our results. We choose to use ML implementations  
789 coming from scikit-learn which is one of the most popular Python ML libraries  
790 at the moment. Its community is active and sensitive to these aspects, we can  
791 assume that if such a problem would exist, it would have been discovered and  
792 fixed quickly.

## 793 7. Related Work

794 **Machine learning and configurable systems.** Machine learning tech-  
795 niques have been widely considered in the literature to learn software configura-  
796 tion spaces [52, 54, 43, 25, 26, 68, 44, 45, 48, 19, 12, 16, 23]. Several works have  
797 proposed to predict the performance of configurations, with several use-cases in  
798 mind for developers and users of configurable systems [60, 12, 63, 62]: the main-  
799 tenance and interpretability of configuration spaces, the exploration of tradeoffs  
800 in the configuration space, the automated specialization of configurable sys-  
801 tems, or simply taking informed decisions when choosing a suited configuration.  
802 The selection of an optimal configuration [48, 19, 47] is also an extensive line  
803 of research. We do not target the problem of finding an optimal configuration  
804 in this article. Though prediction model can be leveraged, more targeted and

805 effective techniques have been proposed to find an optimal configuration [47].  
806 Most of the studies support learning models restrictive to specific static settings,  
807 such that a new prediction model has to be learned from scratch once the en-  
808 vironment changes. The variability of input data exacerbates the problem and  
809 questions the generalization of configuration knowledge, *e.g.*, a configuration is  
810 only optimal for a given input.

811 **Input sensitivity of configurable systems.** Input sensitivity has been  
812 partly considered in some specific research works. Let us take the video encod-  
813 ing [38] as an example: Pereira *et al.* [3] study the effect of sampling training  
814 data from the configuration space on *x264* configuration performance models for  
815 19 input videos on two performance properties. Netflix conducts a large-scale  
816 study for comparing the compression performance of *x264*, *x265*, and *libvpx* [1].  
817 5000 12-second clips from the Netflix catalogue were used, covering a wide range  
818 of genres and signal characteristics. However, only two configurations were con-  
819 sidered and the focus of the study was not on predicting performances. Our  
820 study covers much more inputs, systems, and performance properties. Valov *et*  
821 *al.* [67] proposed a method to transfer the Pareto frontiers (encoding time and  
822 size) of performances across heterogeneous hardware environments. Yet, the in-  
823 puts (video) remain fixed, which is an immediate threat to validity. In fact, this  
824 threat is shared by numerous studies on configurable systems that consider con-  
825 figurations with the same input video (see [52] for the references). In response,  
826 we carefully assess numerous combinations of learning approaches, algorithms,  
827 and input selections to deal with input sensitivity. Input sensitivity is both the  
828 root cause hidden behind the need of input-aware performance models and the  
829 reason why these models may fail at predicting software performance whatever  
830 their input is. If there were no interaction at all between inputs and configu-  
831 rations, a simple performance model for all inputs would suffice. Based on this  
832 work combined with [34], our conjecture is as follows: the more input-sensitive  
833 a software system is, the more difficult (and costly) it is to train an efficient  
834 input-aware performance model.

835 The input sensitivity issue has also been identified — and sometimes dealt  
836 with — in some other domains: SAT solvers [75, 17], compilation [53, 11], data  
837 compression [29], database management [69, 14], cloud computing [15, 36, 12],  
838 *etc.* These works purposely leverage the specifics of their domain. However,  
839 it is unclear how proposed techniques could be adapted to any domain and all  
840 software systems [42]. Thus, We favour a generic, domain-agnostic approach  
841 (*e.g.*, transfer learning) as part of our study. Importantly, most of these works  
842 pursue the objective of optimizing the performance of a software system ac-  
843 cording to a given input (workload). In contrast, we consider the problem of  
844 predicting the performance of any configuration. Our key goal is to investigate  
845 how configuration knowledge can be generalized or transferred among inputs.

846 **Transfer learning.** Transfer learning has been considered for configurable  
847 software systems, with the idea of transferring knowledge across different com-  
848 puting environments *etc.* The promise is to reduce measurements' efforts and  
849 costs over configurations. Jamshidi *et al.* define Learning to Sample (L2S) [26]  
850 that combines an exploitation of the source and an exploration of the target to

851 sample a list of configurations. As many other transfer learning works [4], L2S  
852 is applied to transfer performance of executing environments (*e.g.*, hardware  
853 changes), not input changes. L2S could be adapted as part of transfer learning  
854 (see Figure 2). However, L2S is highly sensitive to the selection of a source (an  
855 input) for a given target (another input). Martin *et al.* develop TEAMs [37], a  
856 transfer learning approach predicting the performance distribution of the Linux  
857 kernel using the measurements of its previous releases. Valov *et al.* showed  
858 that linear models are effective to transfer knowledge across different hardware  
859 environments [68]. However, inputs can significantly alter performance distri-  
860 butions *e.g.*, Pearson correlations can be close to 0 for some pairs of inputs,  
861 systems, and performance properties. There is not necessarily a linear correla-  
862 tion and relationship, as for hardware changes. We assess model shifting as part  
863 of transfer learning. There are many studies in the literature of software engi-  
864 neering applying transfer learning for defect prediction [7, 35, 46, 9, 70]. They  
865 are used for handling a classification problem instead of a regression problem as  
866 in our case. Additionally, while researchers commonly utilize software quality  
867 metrics as predictive features for cross-software defects, our approach differs as  
868 we leverage configuration options to forecast performance. Beyond software sys-  
869 tems, transfer learning is subject to intensive research in many domains (*e.g.*,  
870 image processing, natural language processing) [50, 73, 78]. Different kinds of  
871 data, assumptions, and tasks have been considered. The interplay between con-  
872 figuration options and inputs calls to tackle a regression problem over tabular  
873 data that differ from images or textual content. Some techniques are simply  
874 not applicable in our context. Another specificity of our problem is that there  
875 is this open question on how to select and adapt the source for a given target  
876 (here: a new input fed to a configurable system). Overall, we design transfer  
877 learning techniques that leverage characteristics of inputs and that can operate  
878 over tabular data.

879 In this paper, transfer learning techniques targeting the interplay between  
880 inputs and configurations work best if the source and the target inputs are close  
881 to each other (in terms of performance profiles [11]). To this matter and for  
882 this specific context, the most important part is neither the used ML algorithm  
883 nor the way to transfer the knowledge, but just to associate the right source  
884 input to the target input under prediction. Two insights regarding this finding;  
885 1. to be optimal, transfer learning might require an additional offline effort  
886 (*i.e.*, measuring potential source inputs) to work best, even if the technique is  
887 supposed to be labelled as online, so we can pick the best source input among a  
888 sufficient set of inputs; 2. More than comparing TL techniques with each other,  
889 future efforts should be focusing the optimal association of inputs, how to find  
890 the best source input given the current target input. Our current proposition,  
891 deriving input properties as metrics between inputs to select the best source, can  
892 be seen as an extension of the bellwether effect (*e.g.*, used by BEETLE [32]),  
893 stating there exists a unique source input leading to superior transfer results  
894 whatever the target.

895 **Selection problem.** The automated algorithm selection problem is sub-  
896 ject to intensive research [28, 22, 75, 65]: given a computational problem, a set

897 of algorithms, and a specific problem instance to be solved, the problem is to  
898 determine which of the algorithms can be selected to perform best on that in-  
899 stance. Techniques have substantially improved the State-of-the-Art in solving  
900 many prominent artificial intelligent problems, such as SAT, CSP, QBF, ASP,  
901 or scheduling problems [28]. For instance, SATZilla uses machine learning to  
902 select the most effective algorithm from a portfolio of SAT solvers for a given  
903 SAT formula [75]. There are several differences in our work. First, we target  
904 the problem of predicting the performance of any configuration as opposed to  
905 finding an optimal system. Second, in our case, the set comprises all (valid)  
906 configurations of a single, parameterized, configurable system. In the auto-  
907 mated algorithm setting, the set of algorithms come from different individual  
908 software implementation and systems. As stated in [28] (Section 6), our problem  
909 differs and is still open because (1) the space of valid configurations to select  
910 from is typically very large; (2) learning the mapping from instance features  
911 (*i.e.*, inputs' properties) to configurations is challenging. We precisely address  
912 this problem in this article, considering a large dataset and multiple learning  
913 approaches.

## 914 8. Conclusion

915 Due to the interactions between inputs and configurations, predicting the  
916 performance property of a software configurable system whatever the input data  
917 is non-trivial and yet of practical importance. In particular, performance mod-  
918 els trained on a single input can quickly become inaccurate and useless when  
919 used over other inputs. This lack of generalizability and practicality suggests  
920 to investigate solutions for learning input-aware performance models. In this  
921 article, we empirically evaluated the effectiveness of different learning strategies  
922 (offline learning, supervised online learning, transfer learning) and user personas  
923 when addressing this problem. We leveraged a large dataset comprising 8 soft-  
924 ware systems, hundreds of configurations and inputs, and dozens of performance  
925 properties, spanning a total of 1,941,075 configurations' measurements.

926 Our study empirically proves that measuring the performance of configura-  
927 tions on multiple inputs leads to 1) learning the complexity of predictive perfor-  
928 mance models ; 2) training models which are robust to the change of input data.  
929 Offline learning can build configuration knowledge that pays off and benefits to  
930 online learning when a new input needs to be processed. We emphasize the need  
931 to compute relevant input properties (*e.g.*, video characteristics) as part of the  
932 learning to discriminate the different inputs fed to the software system. As fu-  
933 ture work, we plan to consider input-aware optimization methods. The problem  
934 would differ: instead of transferring the whole performance distribution across  
935 inputs and configurations, optimization pursues the goal of finding a single opti-  
936 mal point, typically through the transfer of some configuration knowledge across  
937 inputs.

938 **Acknowledgements.**

939     This research was funded by the ANR-17-CE25-0010-01 VaryVary project  
940 and the associated Inria/Simula team Resilient Software Science (RESIST\_EA)  
941 <https://gemoc.org/resist/>

942 **References**

- 943 [1] Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne Aaron: A  
944 Large-Scale Comparison of x264, x265, and libvpx – a Sneak Peek. *netflix-*  
945 *study* (2016)
- 946 [2] Alourani, A., Bikas, M., Grechanik, M.: Input-sensitive profiling: A survey.  
947 In: *Advances in Computers*, pp. 31–52. Elsevier (2016)
- 948 [3] Alves Pereira, J., Acher, M., Martin, H., Jézéquel, J.M.: Sampling effect  
949 on performance prediction of configurable systems: A case study. In: *Proc.*  
950 *of ICPE'20*, p. 277–288 (2020)
- 951 [4] Ballesteros, J., Fuentes, L.: Transfer Learning for Multiobjective Optimiza-  
952 tion Algorithms Supporting Dynamic Software Product Lines, p. 51–59.  
953 Association for Computing Machinery, New York, NY, USA (2021). URL  
954 <https://doi.org/10.1145/3461002.3473944>
- 955 [5] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization.  
956 *Journal of machine learning research* **13**(2) (2012)
- 957 [6] Campello, R.J.G.B., Moulavi, D., Sander, J.: Density-Based Clustering  
958 Based on Hierarchical Density Estimates. In: J. Pei, V.S. Tseng, L. Cao,  
959 H. Motoda, G. Xu (eds.) *Advances in Knowledge Discovery and Data Min-*  
960 *ing, Lecture Notes in Computer Science*, pp. 160–172. Springer, Berlin,  
961 Heidelberg (2013). DOI 10.1007/978-3-642-37456-2\_14
- 962 [7] Chen, H., Jing, X.Y., Li, Z., Wu, D., Peng, Y., Huang, Z.: An empirical  
963 study on heterogeneous defect prediction approaches. *IEEE Transactions*  
964 *on Software Engineering* (2020)
- 965 [8] Chen, J., Xu, N., Chen, P., Zhang, H.: Efficient compiler autotuning via  
966 bayesian optimization. In: *Proc. of ICSE'21*, pp. 1198–1209 (2021). DOI  
967 10.1109/ICSE43902.2021.00110
- 968 [9] Chen, J., Yang, Y., Hu, K., Xuan, Q., Liu, Y., Yang, C.: Multiview transfer  
969 learning for software defect prediction. *IEEE Access* **7**, 8901–8916 (2019)
- 970 [10] Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system.  
971 In: *Proceedings of the 22nd ACM SIGKDD International Conference on*  
972 *Knowledge Discovery and Data Mining, KDD '16*, pp. 785–794. ACM,  
973 New York, NY, USA (2016). DOI 10.1145/2939672.2939785. URL [http:](http://doi.acm.org/10.1145/2939672.2939785)  
974 [//doi.acm.org/10.1145/2939672.2939785](http://doi.acm.org/10.1145/2939672.2939785)
- 975 [11] Ding, Y., Ansel, J., Veeramachaneni, K., Shen, X., O'Reilly, U.M., Ama-  
976 rasinghe, S.: Autotuning algorithmic choice for input sensitivity. In: *ACM*  
977 *SIGPLAN Notices*, vol. 50, pp. 379–390. ACM (2015)



- 978 [12] Ding, Y., Pervaiz, A., Carbin, M., Hoffmann, H.: Generalizable and inter-  
979 pretable learning for configuration extrapolation. In: Proceedings of the  
980 29th ACM Joint Meeting on European Software Engineering Conference  
981 and Symposium on the Foundations of Software Engineering, ESEC/FSE  
982 2021, p. 728–740. Association for Computing Machinery, New York, NY,  
983 USA (2021). DOI 10.1145/3468264.3468603. URL [https://doi.org/10.  
984 1145/3468264.3468603](https://doi.org/10.1145/3468264.3468603)
- 985 [13] Dowdy, S.M., Wearden, S.: Statistics for research. Wiley (1983)
- 986 [14] Duan, S., Thummala, V., Babu, S.: Tuning database configuration param-  
987 eters with ituned. *Proc. VLDB Endow.* **2**(1), 1246–1257 (2009). DOI  
988 10.14778/1687627.1687767. URL [https://doi.org/10.14778/1687627.  
989 1687767](https://doi.org/10.14778/1687627.1687767)
- 990 [15] Duarte, F., Gil, R., Romano, P., Lopes, A., Rodrigues, L.: Learning non-  
991 deterministic impact models for adaptation. In: Proc. of SEAMS’18, p.  
992 196–205 (2018). DOI 10.1145/3194133.3194138. URL [https://doi.org/  
993 10.1145/3194133.3194138](https://doi.org/10.1145/3194133.3194138)
- 994 [16] Eggensperger, K., Lindauer, M., Hutter, F.: Neural networks for predicting  
995 algorithm runtime distributions. In: Proceedings of the 27th International  
996 Joint Conference on Artificial Intelligence, IJCAI’18, pp. 1442–1448. AAAI  
997 Press, Stockholm, Sweden (2018)
- 998 [17] Falkner, S., Lindauer, M., Hutter, F.: Spysmac: Automated configuration  
999 and performance analysis of sat solvers. In: Proc. of SAT’15, pp. 215–222  
1000 (2015)
- 1001 [18] Friedman, J.H.: Stochastic gradient boosting. *Computational Statis-  
1002 tics & Data Analysis* **38**(4), 367–378 (2002). DOI [https://doi.org/10.  
1003 1016/S0167-9473\(01\)00065-2](https://doi.org/10.1016/S0167-9473(01)00065-2). URL [https://www.sciencedirect.com/  
1004 science/article/pii/S0167947301000652](https://www.sciencedirect.com/science/article/pii/S0167947301000652). *Nonlinear Methods and Data  
1005 Mining*
- 1006 [19] Fu, W., Menzies, T.: Easy over hard: A case study on deep learning. In:  
1007 Proc. of ESEC-FSE’17, p. 49–60 (2017). DOI 10.1145/3106237.3106256.  
1008 URL <https://doi.org/10.1145/3106237.3106256>
- 1009 [20] Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A.: Revisiting deep  
1010 learning models for tabular data. In: A. Beygelzimer, Y. Dauphin, P. Liang,  
1011 J.W. Vaughan (eds.) *Advances in Neural Information Processing Systems*  
1012 (2021). URL [https://openreview.net/forum?id=i\\_Q1yrOegLY](https://openreview.net/forum?id=i_Q1yrOegLY)
- 1013 [21] Guo, J., Czarnecki, K., Apely, S., Siegmundy, N., Wasowski, A.:  
1014 Variability-aware performance prediction: A statistical learning approach.  
1015 In: Proc. of ASE’13, p. 301–311 (2013). DOI 10.1109/ASE.2013.6693089.  
1016 URL <https://doi.org/10.1109/ASE.2013.6693089>

- 1017 [22] Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based op-  
1018 timization for general algorithm configuration. In: Proc. of LION'05,  
1019 p. 507–523 (2011). DOI 10.1007/978-3-642-25566-3\_40. URL [https://doi.org/10.1007/978-3-642-25566-3\\_40](https://doi.org/10.1007/978-3-642-25566-3_40)  
1020
- 1021 [23] Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime pre-  
1022 diction: Methods & evaluation. Artificial Intelligence **206**, 79–111 (2014).  
1023 DOI 10.1016/j.artint.2013.10.003
- 1024 [24] Iorio, F., Hashemi, A.B., Tao, M., Amza, C.: Transfer learning for cross-  
1025 model regression in performance modeling for the cloud. In: Proc. of Cloud-  
1026 Com'19, pp. 9–18 (2019). DOI 10.1109/CloudCom.2019.00015
- 1027 [25] Jamshidi, P., Siegmund, N., Velez, M., Kästner, C., Patel, A., Agarwal, Y.:  
1028 Transfer learning for performance modeling of configurable systems: An  
1029 exploratory analysis. In: Proc. of ASE'17, p. 497–508 (2017)
- 1030 [26] Jamshidi, P., Velez, M., Kästner, C., Siegmund, N.: Learning to sam-  
1031 ple: Exploiting similarities across environments to learn performance  
1032 models for configurable systems. In: Proc. of ESEC/FSE'18, p. 71–82  
1033 (2018). DOI 10.1145/3236024.3236074. URL <https://doi.org/10.1145/3236024.3236074>  
1034
- 1035 [27] Kendall, M.G.: Rank correlation methods. Griffin (1948)
- 1036 [28] Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated  
1037 algorithm selection: Survey and perspectives. Evolutionary Computa-  
1038 tion **27**(1), 3–45 (2019). DOI 10.1162/evco\_a\_00242. URL [https://doi.org/10.1162/evco\\_a\\_00242](https://doi.org/10.1162/evco_a_00242)  
1039
- 1040 [29] Khavari Tavana, M., Sun, Y., Bohm Agostini, N., Kaeli, D.: Exploiting  
1041 adaptive data compression to improve performance and energy-efficiency  
1042 of compute workloads in multi-gpu systems. In: Proc. of IPDPS'19, pp.  
1043 664–674 (2019). DOI 10.1109/IPDPS.2019.00075
- 1044 [30] Knüppel, A., Thüm, T., Pardylla, C.I., Schaefer, I.: Understanding pa-  
1045 rameters of deductive verification: An empirical investigation of key. In:  
1046 J. Avigad, A. Mahboubi (eds.) Interactive Theorem Proving - 9th Inter-  
1047 national Conference, ITP 2018, Held as Part of the Federated Logic  
1048 Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings, *Lec-  
1049 ture Notes in Computer Science*, vol. 10895, pp. 342–361. Springer (2018).  
1050 DOI 10.1007/978-3-319-94821-8\_20. URL [https://doi.org/10.1007/978-3-319-94821-8\\_20](https://doi.org/10.1007/978-3-319-94821-8_20)  
1051
- 1052 [31] Krause, A., Golovin, D.: Submodular function maximization. Tractability  
1053 **3**, 71–104 (2014)
- 1054 [32] Krishna, R., Nair, V., Jamshidi, P., Menzies, T.: Whence to learn? trans-  
1055 ferring knowledge in configurable systems using beetle. IEEE Transactions

- 1056 on Software Engineering **47**(12), 2956–2972 (2021). DOI 10.1109/TSE.  
1057 2020.2983927
- 1058 [33] Larsson, H., Taghia, J., Moradi, F., Johnsson, A.: Source selection in  
1059 transfer learning for improved service performance predictions. In: Proc.  
1060 of Networking’21, pp. 1–9 (2021). DOI 10.23919/IFIPNetworking52078.  
1061 2021.9472818
- 1062 [34] Lesoil, L., Acher, M., Blouin, A., Jézéquel, J.M.: Input Sensitivity on the  
1063 Performance of Configurable Systems: An Empirical Study. *Journal of Sys-*  
1064 *tems and Software* (2023). URL <https://hal.inria.fr/hal-03476464>
- 1065 [35] Li, Z., Jing, X.Y., Wu, F., Zhu, X., Xu, B., Ying, S.: Cost-sensitive transfer  
1066 kernel canonical correlation analysis for heterogeneous defect prediction.  
1067 *ASE* **25**(2), 201–245 (2018)
- 1068 [36] Liu, F., Miniskar, N.R., Chakraborty, D., Vetter, J.S.: Deffe: A data-  
1069 efficient framework for performance characterization in domain-specific  
1070 computing. In: Proc. of CF’20, p. 182–191 (2020). DOI 10.1145/3387902.  
1071 3392633. URL <https://doi.org/10.1145/3387902.3392633>
- 1072 [37] Martin, H., Acher, M., Lesoil, L., Jezequel, J.M., Khelladi, D.E., Pereira,  
1073 J.A.: Transfer learning across variants and versions : The case of linux  
1074 kernel size. *IEEE Transactions on Software Engineering* **1**, 1–1 (2021).  
1075 DOI 10.1109/TSE.2021.3116768
- 1076 [38] Maxiaguine, A., Yanhong Liu, Chakraborty, S., Wei Tsang Ooi: Identifying  
1077 ”representative” workloads in designing mpsoC platforms for media  
1078 processing. In: 2nd Workshop on Embedded Systems for Real-Time  
1079 Multimedia, 2004. ESTImedia 2004., pp. 41–46 (2004). URL <https://ieeexplore.ieee.org/document/1359702>
- 1081 [39] McInnes, L., Healy, J., Astels, S.: hdbscan: Hierarchical density based  
1082 clustering. *The Journal of Open Source Software* **2**(11) (2017). DOI  
1083 10.21105/joss.00205. URL <https://doi.org/10.21105%2Fjoss.00205>
- 1084 [40] Molnar, C.: *Interpretable Machine Learning*. Lulu.com, München, Bayern,  
1085 Deutschland (2020)
- 1086 [41] Mühlbauer, C.S., Sattler, F., Siegmund, N.: Analyzing the impact of  
1087 workloads on modeling the performance of configurable software systems.  
1088 In: Proceedings of the International Conference on Software Engineering  
1089 (ICSE), IEEE (2023)
- 1090 [42] Mühlbauer, S., Sattler, F., Kaltenecker, C., Dorn, J., Apel, S., Siegmund,  
1091 N.: Analyzing the impact of workloads on modeling the performance of  
1092 configurable software systems. In: Proceedings of the 45th International  
1093 Conference on Software Engineering. IEEE (2023)

- 1094 [43] Nair, V., Krishna, R., Menzies, T., Jamshidi, P.: Transfer learning with  
1095 bellwethers to find good configurations. CoRR **abs/1803.03900** (2018).  
1096 URL <http://arxiv.org/abs/1803.03900>
- 1097 [44] Nair, V., Menzies, T., Siegmund, N., Apel, S.: Using bad learners to find  
1098 good configurations. In: Proc. of ESEC/FSE'17, pp. 257–267 (2017)
- 1099 [45] Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S.: Finding faster con-  
1100 figurations using flash. IEEE Transactions on Software Engineering (2018)
- 1101 [46] Nam, J., Fu, W., Kim, S., Menzies, T., Tan, L.: Heterogeneous defect  
1102 prediction. IEEE Transactions on Software Engineering **44**(9), 874–896  
1103 (2017)
- 1104 [47] OH, J., Batory, D., HERADIO, R.: Finding near-optimal configurations in  
1105 colossal spaces with statistical guarantees. TOSEM (2023)
- 1106 [48] Oh, J., Batory, D., Myers, M., Siegmund, N.: Finding near-optimal con-  
1107 figurations in product lines by random sampling. In: Proceedings of  
1108 the 2017 11th Joint Meeting on Foundations of Software Engineering,  
1109 ESEC/FSE 2017, p. 61–71. Association for Computing Machinery, New  
1110 York, NY, USA (2017). DOI 10.1145/3106237.3106273. URL <https://doi.org/10.1145/3106237.3106273>
- 1112 [49] Oshiro, T.M., Perez, P.S., Baranauskas, J.A.: How many trees in a random  
1113 forest? In: P. Perner (ed.) Machine Learning and Data Mining in Pattern  
1114 Recognition, pp. 154–168. Springer Berlin Heidelberg, Berlin, Heidelberg  
1115 (2012)
- 1116 [50] Pan, S.J., Yang, Q.: A survey on transfer learning. TKDE **22**(10), 1345–  
1117 1359 (2009)
- 1118 [51] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B.,  
1119 Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.:  
1120 Scikit-learn: Machine learning in python. the Journal of machine Learning  
1121 research **12**, 2825–2830 (2011)
- 1122 [52] Pereira, J.A., Acher, M., Martin, H., Jézéquel, J.M., Botterweck, G., Ven-  
1123 tresque, A.: Learning software configuration spaces: A systematic litera-  
1124 ture review. JSS p. 111044 (2021). DOI [https://doi.org/10.1016/j.jss.2021.](https://doi.org/10.1016/j.jss.2021.111044)  
1125 111044. URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/S0164121221001412)  
1126 [S0164121221001412](https://www.sciencedirect.com/science/article/pii/S0164121221001412)
- 1127 [53] Plotnikov, D., Melnik, D., Vardanyan, M., Buchatskiy, R., Zhuykov, R.,  
1128 Lee, J.H.: Automatic tuning of compiler optimizations and analysis of  
1129 their impact. Procedia Computer Science **18**, 1312–1321 (2013). DOI  
1130 10.1016/j.procs.2013.05.298. URL [https://doi.org/10.1016/j.procs.](https://doi.org/10.1016/j.procs.2013.05.298)  
1131 [2013.05.298](https://doi.org/10.1016/j.procs.2013.05.298)

- 1132 [54] Quinton, C., Vierhauser, M., Rabiser, R., Baresi, L., Grünbacher, P.,  
1133 Schuhmayer, C.: Evolution in dynamic software product lines. *Journal*  
1134 *of Software: Evolution and Process* p. e2293 (2020)
- 1135 [55] Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier method-  
1136 ology. *IEEE transactions on systems, man, and cybernetics* **21**(3), 660–674  
1137 (1991)
- 1138 [56] Schreiber, J., Bilmes, J., Noble, W.S.: Apricot: Submodular selection for  
1139 data summarization in Python. *Journal of Machine Learning Research*  
1140 **21**(161), 1–6 (2020)
- 1141 [57] Seber, G.A., Lee, A.J.: *Linear regression analysis*, vol. 329. John Wiley &  
1142 Sons, North America (2012)
- 1143 [58] Shwartz-Ziv, R., Armon, A.: Tabular data: Deep learning is not all you  
1144 need. *Information Fusion* **81**, 84–90 (2022). DOI 10.1016/j.inffus.2021.11.  
1145 011
- 1146 [59] Siegel, S.: *Nonparametric statistics for the behavioral sciences*. *The Journal*  
1147 *of Nervous and Mental Disease* **125**, 497 (1956)
- 1148 [60] Siegmund, N., Grebhahn, A., Kästner, C., Apel, S.: Performance-influence  
1149 models for highly configurable systems. In: *ESEC/FSE’15* (2015). URL  
1150 <https://doi.org/10.1145/2786805.2786845>
- 1151 [61] Sinha, U., Cashman, M., Cohen, M.B.: Using a genetic algorithm to opti-  
1152 mize configurations in a data-driven application. In: A. Aleti, A. Panichella  
1153 (eds.) *Search-Based Software Engineering - 12th International Sympo-*  
1154 *sium, SSBSE 2020, Bari, Italy, October 7-8, 2020, Proceedings, Lecture*  
1155 *Notes in Computer Science*, vol. 12420, pp. 137–152. Springer (2020).  
1156 DOI 10.1007/978-3-030-59762-7\_10. URL [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-030-59762-7_10)  
1157 [978-3-030-59762-7\\_10](https://doi.org/10.1007/978-3-030-59762-7_10)
- 1158 [62] Temple, P., Acher, M., Jezequel, J.M., Barais, O.: Learning contextual-  
1159 variability models. *IEEE Software* **34**(6), 64–70 (2017)
- 1160 [63] Temple, P., Acher, M., Jézéquel, J.M., Noel-Baron, L., Galindo, J.A.:  
1161 Learning-Based Performance Specialization of Configurable Systems. Re-  
1162 search report, IRISA, Inria Rennes ; University of Rennes 1 (2017). URL  
1163 <https://hal.archives-ouvertes.fr/hal-01467299>
- 1164 [64] Thirey, B., Hickman, R.: Distribution of euclidean distances be-  
1165 tween randomly distributed gaussian points in n-space. *arXiv preprint*  
1166 *arXiv:1508.02238* (2015)
- 1167 [65] Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka:  
1168 Combined selection and hyperparameter optimization of classification al-  
1169 gorithms. In: *Proc. of KDD’13*, p. 847–855 (2013)

- 1170 [66] Valov, P., Guo, J., Czarnecki, K.: Empirical comparison of regression  
1171 methods for variability-aware performance prediction. In: Proceedings of  
1172 the 19th International Conference on Software Product Line, SPLC '15,  
1173 p. 186–190. Association for Computing Machinery, New York, NY, USA  
1174 (2015). DOI 10.1145/2791060.2791069. URL [https://doi.org/10.1145/  
1175 2791060.2791069](https://doi.org/10.1145/2791060.2791069)
- 1176 [67] Valov, P., Guo, J., Czarnecki, K.: Transferring pareto frontiers across het-  
1177 erogeneous hardware environments. In: Proceedings of the ACM/SPEC  
1178 International Conference on Performance Engineering, ICPE '20, p. 12–23.  
1179 Association for Computing Machinery, New York, NY, USA (2020). DOI  
1180 10.1145/3358960.3379127. URL [https://doi.org/10.1145/3358960.  
1181 3379127](https://doi.org/10.1145/3358960.3379127)
- 1182 [68] Valov, P., Petkovich, J.C., Guo, J., Fischmeister, S., Czarnecki, K.: Trans-  
1183 ferring performance prediction models across different hardware platforms.  
1184 In: Proceedings of the 8th ACM/SPEC on International Conference on  
1185 Performance Engineering, ICPE '17, p. 39–50. Association for Computing  
1186 Machinery, New York, NY, USA (2017). DOI 10.1145/3030207.3030216.  
1187 URL <https://doi.org/10.1145/3030207.3030216>
- 1188 [69] Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B.: Automatic database  
1189 management system tuning through large-scale machine learning. In: Pro-  
1190 ceedings of the 2017 ACM International Conference on Management of  
1191 Data, SIGMOD '17, p. 1009–1024. Association for Computing Machin-  
1192 ery, New York, NY, USA (2017). DOI 10.1145/3035918.3064029. URL  
1193 <https://doi.org/10.1145/3035918.3064029>
- 1194 [70] Wang, A., Zhang, Y., Wu, H., Jiang, K., Wang, M.: Few-shot learning  
1195 based balanced distribution adaptation for heterogeneous defect prediction.  
1196 *IEEE Access* **8**, 32989–33001 (2020)
- 1197 [71] Wang, Y., Inguva, S., Adsumilli, B.: Youtube ugc dataset for video com-  
1198 pression research. 2019 IEEE 21st International Workshop on Multimedia  
1199 Signal Processing (MMSP) (2019). DOI 10.1109/mmisp.2019.8901772. URL  
1200 <http://dx.doi.org/10.1109/MMSP.2019.8901772>
- 1201 [72] Wei, H., Zhou, S., Yang, T., Zhang, R., Wang, Q.: Elastic resource man-  
1202 agement for heterogeneous applications on paas. In: Proceedings of the 5th  
1203 Asia-Pacific Symposium on Internetware, Internetware '13. Association for  
1204 Computing Machinery, New York, NY, USA (2013). DOI 10.1145/2532443.  
1205 2532451. URL <https://doi.org/10.1145/2532443.2532451>
- 1206 [73] Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning.  
1207 *Journal of Big data* **3**(1), 9 (2016)
- 1208 [74] Xu, B., Zhao, D., Jia, K., Zhou, J., Tian, J., Xiang, J.: Cross-project aging-  
1209 related bug prediction based on joint distribution adaptation and improved

- 1210 subclass discriminant analysis. In: 2020 IEEE 31st International Symposi-  
1211 sium on Software Reliability Engineering (ISSRE), pp. 325–334 (2020).  
1212 DOI 10.1109/ISSRE5003.2020.00038
- 1213 [75] Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-  
1214 based algorithm selection for sat. *Journal of artificial intelligence research*  
1215 **32**, 565–606 (2008)
- 1216 [76] Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., Talwadker, R.: Hey,  
1217 you have given me too many knobs!: Understanding and dealing with over-  
1218 designed configuration in system software. In: *Proceedings of the 2015*  
1219 *10th Joint Meeting on Foundations of Software Engineering*, pp. 307–319  
1220 (2015). DOI 10.1145/2786805.2786852
- 1221 [77] Zhang, J., Liu, Y., Zhou, K., Li, G., Xiao, Z., Cheng, B., Xing, J., Wang,  
1222 Y., Cheng, T., Liu, L., Ran, M., Li, Z.: An end-to-end automatic cloud  
1223 database tuning system using deep reinforcement learning. In: *Proceedings*  
1224 *of the 2019 International Conference on Management of Data, SIGMOD*  
1225 *'19*, p. 415–432. Association for Computing Machinery, New York, NY,  
1226 USA (2019). DOI 10.1145/3299869.3300085. URL [https://doi.org/10.](https://doi.org/10.1145/3299869.3300085)  
1227 [1145/3299869.3300085](https://doi.org/10.1145/3299869.3300085)
- 1228 [78] Zhang, W., Deng, L., Zhang, L., Wu, D.: Overcoming negative transfer: A  
1229 survey. *arXiv preprint arXiv:2009.00909* (2020)