The Journal of Systems & Software 182 (2021) 111044

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Learning software configuration spaces: A systematic literature review[☆]

Juliana Alves Pereira [a,b,*], Mathieu Acher [a], Hugo Martin [a], Jean-Marc Jézéquel [a], Goetz Botterweck [c], Anthony Ventresque [d]

[a] Univ Rennes, Inria, CNRS, IRISA, France
[b] Pontifical Catholic University of Rio de Janeiro, Brazil
[c] Trinity College Dublin, Lero–The Irish Software Research Centre, Ireland
[d] University College Dublin, Ireland

ARTICLE INFO

Article history:
Received 7 June 2019
Received in revised form 15 April 2021
Accepted 12 July 2021
Available online 20 August 2021

Keywords:
Systematic literature review
Software product lines
Machine learning
Configurable systems

ABSTRACT

Most modern software systems (operating systems like Linux or Android, Web browsers like Firefox or Chrome, video encoders like ffmpeg, x264 or VLC, mobile and cloud applications, etc.) are highly configurable. Hundreds of configuration options, features, or plugins can be combined, each potentially with distinct functionality and effects on execution time, security, energy consumption, etc. Due to the combinatorial explosion and the cost of executing software, it is quickly impossible to exhaustively explore the whole configuration space. Hence, numerous works have investigated the idea of learning it from a small sample of configurations' measurements. The pattern "sampling, measuring, learning" has emerged in the literature, with several practical interests for both software developers and end-users of configurable systems. In this systematic literature review, we report on the different application objectives (*e.g.*, performance prediction, configuration optimization, constraint mining), use-cases, targeted software systems, and application domains. We review the various strategies employed to gather a representative and cost-effective sample. We describe automated software techniques used to measure functional and non-functional properties of configurations. We classify machine learning algorithms and how they relate to the pursued application. Finally, we also describe how researchers evaluate the quality of the learning process. The findings from this systematic review show that the potential application objective is important; there are a vast number of case studies reported in the literature related to particular domains or software systems. Yet, the huge variant space of configurable systems is still challenging and calls to further investigate the synergies between artificial intelligence and software engineering.

1. Introduction

End-users, system administrators, software engineers, and scientists have at their disposal thousands of options (a.k.a. features or parameters) to configure various kinds of software systems in order to fit their functional and non-functional needs (execution time, output quality, security, energy consumption, etc.). It is now ubiquitous that software comes in many variants and is highly configurable through conditional compilations, command-line options, runtime parameters, configuration files, or plugins. Software product lines (SPLs), software generators, dynamic system, self-adaptive systems, variability-intensive systems are well studied in the literature and enter in this class of configurable software systems (Svahnberg et al., 2005; Pohl et al., 2005; Apel et al., 2013; Sayagh et al., 2018; Benavides et al., 2010; Cashman et al., 2018; Hallsteinsen et al., 2008; Morin et al., 2009).

From an abstract point of view, a software configuration is simply a combination of options' values. Though customization is highly desirable, it introduces an enormous complexity due to the combinatorial explosion of possible variants. For example, the Linux kernel has 15,000+ options and most of them can have 3 values: "yes", "no", or "module". Without considering the presence of constraints to avoid some combinations of options, there may be $3^{15,000}$ possible variants of Linux — the estimated number of atoms in the universe is 10^{80} and is already reached with 300 Boolean options. Though Linux is an extreme case, many software systems or projects exhibit a very large configuration space; this might bring several challenges.

On the one hand, developers struggle to maintain, understand, and test configuration spaces since they can hardly analyze or execute all possible variants. According to several studies (Halin

[☆] Editor: Raffaela Mirandola.
[*] Corresponding author.
E-mail address: jpereira@inf.puc-rio.br (J.A. Pereira).

https://doi.org/10.1016/j.jss.2021.111044
0164-1212/© 2021 Elsevier Inc. All rights reserved.

et al., 2019; Sayagh et al., 2018), the flexibility brought by variability is expensive as configuration failures represent one of the most common types of software failures. Configuration failures is an "undesired effect observed in the system's delivered service" that may occur due to a specific combination of options' values (configurations) (Mathur, 2008). On the other hand, end-users fear software variability and stick to default configurations (Xu et al., 2015; Zheng et al., 2007) that may be sub-optimal (*e.g.*, the software system will run very slowly) or simply inadequate (*e.g.*, the quality of the output will be unsuitable).

Since it is hardly possible to fully explore all software configurations, the use of machine learning techniques is a quite natural and appealing approach. The basic idea is to learn out of a *sample* of configurations' observations and hopefully generalize to the whole configuration space. There are several applications ranging from performance prediction, configuration optimization, software understanding to constraint mining (*i.e.*, extraction of variability rules) – we will give a more exhaustive list in this literature review. For instance, end-users of x264 (a configurable video encoder) can estimate in advance the execution time of the command-line `X264 --no_cabac --no_fast_pskip --rc_lookahead 60 --ref 5 -o vid.264 vid.y4m` (see Fig. 1), since a machine learning model has been crafted to predict the performance of configurations. End-users may want to use the fastest configuration or know all configurations that meet an objective (*e.g.*, encoding time should be less than 10 s). Developers of x264 can be interested in understanding the effects of some options and how options interact.

For all these use-cases, a pattern has emerged in the scientific literature: *"sampling, measuring, learning"*. The basic principle is that a procedure is able to learn out of a sample of configurations' measurements (see Fig. 1). Specifically, many software configuration problems can actually be framed as statistical machine learning problems under the condition a sample of configurations' observations is available. For example, the prediction of the performance of individual configurations can be formulated as a regression problem; appropriate learning algorithms (*e.g.*, CART) can then be used to predict the performance of untested, new configurations. In this respect, it is worth noticing the dual use of the term *feature* in the software or machine learning fields: features either refer to software features (*a.k.a.* configuration options) or to variables a regressor aims to relate. A way to reconcile and visualize both is to consider a configuration matrix as depicted in Fig. 1. In a configuration matrix, each row describes a configuration together with observations/values of each feature and performance property. In the example of Fig. 1, the first configuration has the feature `no_cabac` set to False value and the feature `ref` set to 9 value while the encoding time performance value is 3.1876 s. We can use a sample of configurations (*i.e.*, a set of measured configurations) to train a machine learning model (a regressor) with predictive variables being command-line parameters of x264. Unmeasured configurations could then be predicted. Even for large-scale systems like the Linux Kernel, the same process of "sampling, measuring, learning" can be followed (see, e.g. Acher et al. (2019a,b)). Some additional steps are worth exploring (like feature engineering prior to learning) while techniques for sampling and learning should be adapted to scale at its complexity, but the general process remains applicable.

Learning software configuration spaces is, however, not a pure machine learning problem and there are a number of specific challenges to address at the intersection of software engineering and artificial intelligence. For instance, the sampling phase involves a number of difficult activities: (1) picking configurations that are valid and conform to constraints among options – one needs to resolve a satisfiability problem; (2) instrumenting the executions and observations of software for a variety of configurations – it can have an important computational cost and is hard to engineer especially when measuring non-functional aspects of software; (3) meanwhile, we expect that the sample is representative to the whole population of valid configurations otherwise the learning algorithm may hardly generalize to the whole configuration space. The general problem is to find the right strategy to decrease the cost of labeling software configurations while minimizing prediction errors. From an empirical perspective, one can also wonder to what extent learning approaches are effective for real-world software systems present in numerous domains.

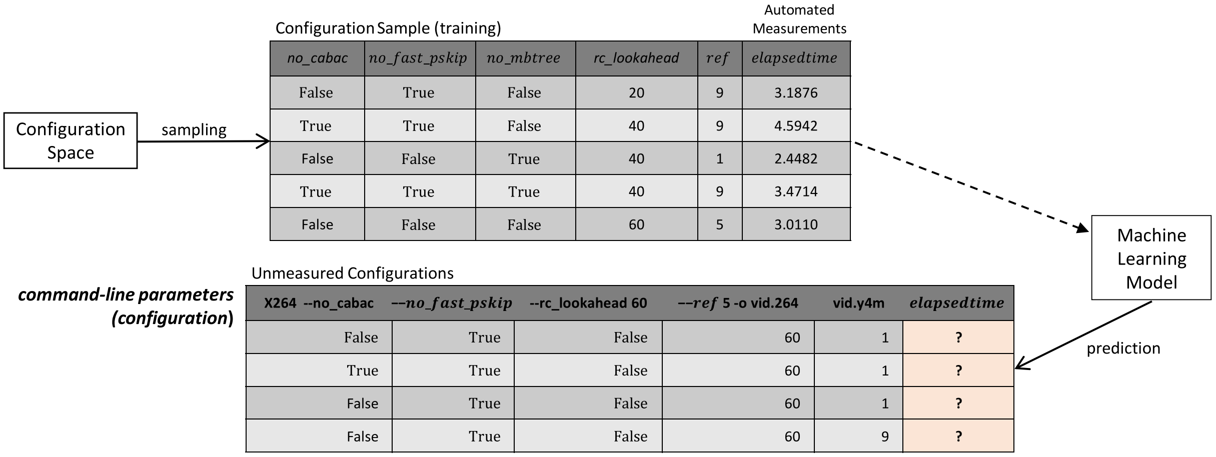
While several studies have covered different aspects of configurable systems over the last years, there has been no secondary study (such as systematic literature reviews) that identifies and catalogs individual contributions for machine learning configuration spaces. Thus, there is no clear consensus on what techniques are used to support the process, including which quantitative and qualitative properties are considered and how they can be measured and evaluated, as well as how to select a significant sample of configurations and what is an ideal sample size. This stresses the need for a secondary study to build knowledge from combining findings from different approaches and present a complete overview of the progress made in this field. To achieve this aim, we conduct a *Systematic Literature Review* (SLR) (Kitchenham and Charters, 2007) to identify, analyze and interpret all available important research in this domain. We systematically review research papers in which the process of sampling, measuring, and learning configuration spaces occurs — more details about our research methodology are given in Section 2. Specifically, we aim of synthesizing evidence to answer the following four research questions:

- *RQ1. What are the concrete applications of learning software configuration spaces?*
- *RQ2. Which sampling methods and learning techniques are adopted when learning software configuration spaces?*
- *RQ3. Which techniques are used to gather measurements of functional and non-functional properties of configurations?*
- *RQ4. How are learning-based techniques validated?*

To address *RQ1*, we analyze the application objective of the study (*i.e.*, why they apply learning-based techniques). It would allow us to assess whether the proposed approaches are applicable. With respect to *RQ2*, we systematically investigate which sampling methods and learning techniques are used in the literature for exploring the SPL configuration space. With respect to *RQ3*, we give an in-depth view of how each study measures a sample of configurations. In addition, *RQ4* follows identifying which sampling design and evaluation metrics are used for evaluation.

By answering these questions, we make the following five contributions:

1. We identified six main different application areas: *pure prediction*, *interpretability*, *optimization*, *dynamic configuration*, *evolution*, and *mining constraints*.
2. We provide a framework classification of four main stages used for learning: *Sampling*, *Measuring*, *Learning*, *Validation*.
3. We describe 23 high-level sampling methods, 5 measurement strategies, 64 learning techniques, and 50 evaluation metrics used in the literature. As case studies, we identify 95 configurable systems targeting several domains, and functional and non-functional properties. We relate and discuss the learning and validation techniques with regard to their application objective.
4. We identify a set of open challenges faced by the current approaches, in order to guide researchers and practitioners to use and build appropriate solutions.


Configuration Space
sampling
Configuration Sample (training)
Automated Measurements
Machine Learning Model
prediction
Unmeasured Configurations
command-line parameters (configuration)

no_cabac	*no_fast_pskip*	*no_mbtree*	*rc_lookahead*	*ref*	*elapsedtime*
False	True	False	20	9	3.1876
True	True	False	40	9	4.5942
False	False	True	40	1	2.4482
True	True	True	40	9	3.4714
False	False	False	60	5	3.0110

X264 --no_cabac	--*no_fast_pskip*	--rc_lookahead 60	--*ref* 5 -o vid.264	vid.y4m	*elapsedtime*
False	True	False	60	1	?
True	True	False	60	1	?
False	True	False	60	1	?
False	True	False	60	9	?

Fig. 1. Features, configurations, sample, measurements, and learning.

5. We build a Web repository (Pereira et al., 2019) to make our SLR results publicly available for the purpose of reproducibility and extension.

Overall, the findings of this SLR reveal that there is a significant body of work specialized in learning software configurable systems with an important application in terms of software technologies, application domains, or goals. There is a wide variety in the considered sampling or learning algorithms as well as in the evaluation process, mainly due to the considered subject systems and application objectives. Practitioners and researchers can benefit from the findings reported in this SLR as a reference when they select a learning technique for their own settings. To this end, this review provides a classification and catalog of specialized techniques in this field.

The rest of the paper is structured as follows. In Section 2, we describe the research protocol used to conduct the SLR. In Section 3, we categorize a sequence of key learning stages used by the ML state-of-the-art literature to explore highly configurable systems. In Section 4, we discuss the research questions. In Section 5, we discuss the current research themes in this field and present the open challenges that need attention in the future. In Section 6, we discuss the threats to the validity of our SLR. In Section 7, we describe similar secondary studies and indicate how our literature review differs from them. Finally, in Section 8, we present the conclusions of our work.

2. The review methodology

We followed the SLR guidelines by Kitchenham and Charters (2007) to systematically investigate the use of learning techniques for exploring the SPL configuration space. In this section, we present the SLR methodology that covers two main phases: *planning the review* and *conducting the review*. The paper selection process is shown in Fig. 2. Next, we report the details about each phase so that readers can assess their rigor and completeness, and reproduce our findings.

2.1. Planning the review

For identifying the candidate primary studies of Fig. 2, we first defined our SLR scope (i.e., identification of the need for a review and specification of the research questions). Then, we developed a review protocol.

The need for a systematic review. The main goal of this SLR is to systematically investigate and summarize the state-of-the-art of the research concerning learning techniques in the context of software configurable systems. The purpose of conducting this SLR has partially been addressed in the introduction and was motivated by the lack of a systematic study carried on this topic. According to Kitchenham and Charters (2007), an SLR should provide a valuable overview of the status of the field to the community through the summarization of existing empirical evidence supported by current scientific studies. The outcomes of such an overview can identify whether, or under what conditions, the proposed learning approaches can support various use-cases around configurable systems and be practically adopted (*e.g.*, for which context a specific learning technique is much suitable). By mean of this outcome, we can detect the limitations in current approaches to properly suggest areas for further investigation.

The research questions. The goal of this SLR is to answer the following main research question: *What studies have been reported in the literature on learning software configuration spaces since the introduction of Software Product Lines in the early 1990s (Kang et al., 1990) to date (2019)?* However, this question is too broad, so we derived the four sub-questions defined in Section 1. *RQ1* classifies the papers with regards to their application objective, i.e., for which particular task the approach is suited and useful. We can group studies into similar categories for comparison. It is also of interest to identify the practical motivations behind learning approaches. We verified whether the authors indicated a specific application for their approach; otherwise, we classified the approach as pure prediction. *RQ2–RQ4* seek to understand key steps of the learning process. *RQ2* reviews the set of sampling methods and learning-based techniques used in the literature. *RQ3* describes which subject software systems, application domains, and functional and non-functional properties of configurations are measured and how the measurement process is conducted. *RQ4* aims to characterize the evaluation process used by the researchers, including the sample design and supported evaluation metric(s). Finally, addressing these questions will allow us to identify trends and challenges in the current state-of-the-art approaches, as well as analysis their maturity to summarize our findings and propose future works.

The review protocol. We searched for all relevant papers published up to May 31st, 2019. The search process involved the use of 5 scientific digital libraries[1]: IEEE Xplore Digital Library,[2]

[1] We decided not to use Google Scholar due to search engine limitations, such as the very strict size of the search string.

[2] http://ieee.org/ieeexplore.

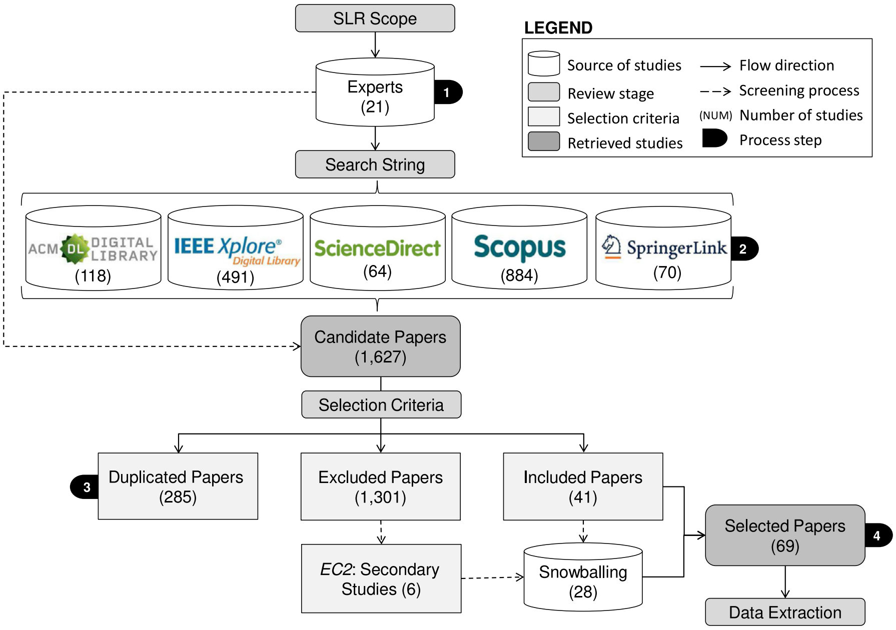


Fig. 2. Flow of the paper selection process: papers retrieved from each digital library and details of the selection phases.

ACM Digital Library,[3] Science Direct,[4] Springer-link,[5] and Scopus[6] (see Fig. 2). These search engines were selected because they are known as the top five preferred on-line databases in the software engineering literature (Hoda et al., 2017). We restricted the search to publication titles and abstracts by applying the selection criteria specified in Section 2.2. However, the library Springer-link only enables a full-text search. Thus, we first used the full-text option to generate an initial set of papers (the results were stored in a .bib file). Then, we created a script to perform an expert search in the title and abstract over these results.

Each author of this paper had specific roles when performing this SLR. The first author applied the search string to the scientific databases and exported the results (*i.e.*, detailed information about the candidate papers) into a spreadsheet. After this stage, papers were selected based on a careful reading of the titles and abstracts (and if necessary checking the introduction and conclusion). Each identified candidate paper in accordance with the selection criteria defined in Section 2.2 were identified as potentially relevant. When Pereira decided that a paper was not relevant, she provided a short rationale why the paper should not be included in the study. In addition, another researcher checked each excluded paper at this stage. To minimize potential biases introduced into the selection process, any disagreement between researchers were put up for discussion between all authors until a consensus agreement was obtained. This step was done in order to check that all relevant papers were selected.

The search in such databases is known to be challenging due to different search limitations, *e.g.* different ways of constructing the search string. Thus, apart from the automatic search, we also consider the use of snowballing (Wohlin, 2014) as a complementary approach. Through snowballing, we searched for additional relevant primary studies by following the references from all preliminary selected studies (plus excluded secondary studies). As we published some works related to the topic of this literature review, we used our knowledge and expertise to complement the pool of relevant papers.

During the data extraction stage, each paper was assigned to one researcher. Pereira coordinated the allocation of researchers to tasks based on the availability of each researcher. The researcher responsible for extracting the data of a specific selected paper, applied the snowballing technique to the corresponding paper. Pereira applied the snowballing technique for excluded secondary studies. Each primary study was then assigned to another researcher for review. Once all the primary studies were reviewed, the extracted data was compared. Whenever there were any discrepancies either about the data reported or about the list of additional selected papers derived from the snowballing process, we again resolved the problem through discussions among all authors.

2.2. Conducting the review

The stages involved in *conducting the review* are: definition of the search string, specification of the selection criteria, and specification of the data extraction process.

As a first step, we used our expertise in the field to obtain an initial pool of relevant studies. Based on this effort, we defined the search string.

The search string. According to Kitchenham and Charters (2007) there is no silver bullet for identifying good search strings since, very often, the terminology used in the field is not standardized. When using broad search terms, a large number of irrelevant papers may be found in the search which makes the screening process very challenging. The search string used in this review was first initiated by selecting an initial list of relevant articles based on our expertise in the field.

We identified in the title and abstract the major terms to be used for systematic searching the primary studies. Then, we searched for synonyms related to each major term. Next, we performed several test searches with alternative links between keywords through the different digital libraries. The results from the test searches were continuously discussed among the authors to refine the search string until we were fully satisfied with the capability of the string to detect as much of the initial set of relevant publications as possible. Following this iterative strategy

[3] http://dl.acm.org.
[4] http://www.sciencedirect.com.
[5] http://link.springer.com.
[6] http://www.scopus.com.

Table 1
Keywords used to build the search strings.

Term	Keywords
Product line	product line, configurable (system, software), software configurations, configuration of a software, feature (selection, configuration)
Learning	learning techniques, (machine, model, statistical) learning
Performance prediction	performance (prediction, model, goal), (software, program, system) performance, (prediction of, measure) non-functional properties
Predict	predict, measure, transfer learning, optimal (configuration, variant), adaptation rules, constraints, context
Medicine	gene, medicine, disease, patient, biology, diagnosis, molecular, health, brain, biomedical

and after a series of test executions and reviews, we obtained a set of search terms and keywords (see Table 1).

Specifically, Table 1 shows the *term* we are looking for and related synonyms that we considered as *keywords* in our search. Keywords associated with `Product Line` allow us to include studies that focus on configurable systems. By combining keywords associated to `Learning` and `Performance Prediction`, we can find studies that focus on the use of learning-based techniques for exploring the variability space. In addition, keywords associated with `Predict` (most specific term) allow us to focus on the application objective of such studies. We decided to include the keywords associated with `Predict` so as to identify the context of the study and have a more restricted number of primary studies. Otherwise, the keywords `(Product Line AND (Learning OR Performance Prediction))` return a broad number of studies, *e.g.* studies addressing the use of learning techniques for product line testing or configuration guidance. In addition, we used keywords from `Medicine` to exclude studies in this field from our search. We decided to filter out these studies once there are many papers in this field using also the term configuration, e.g. for gene and patient characteristics. The final result is the following search string:

(Product Line `AND` (Learning `OR` Performance Prediction)
`AND` Predict) `AND NOT` Medicine

The terms `Product Line`, `Learning`, `Performance Prediction`, and `Predict` are represented as a disjunction of the keywords in Table 1. The search string format we used was slightly modified to meet the search engine requirements of the different scientific databases. For example, the scientific library Science Direct limits the size of the search string. Thus, when searching in this library, we had to split the search string to generate an initial set of papers. Then, we created a script to perform an expert search over this initial set of papers and filter just the relevant ones from these subsets, *i.e.* we made every effort to ensure that the search strings used were logically and semantically equivalent to the original string in Table 1. The detailed search strings used in each digital search engine and additional scripts are provided in the Web supplementary material (Pereira et al., 2019).

As a second step, we applied the search string to the scientific digital libraries. Fig. 2 shows the number of papers obtained from each library. At the end of step 2, the initial search from all sources resulted in a total of 1,627 candidate papers, which includes the 21 papers from our initial pool of relevant studies. As a third step, we removed all duplicated papers (285) and we carried out the selection process at the content level.

The selection criteria. We selected the studies using a set of selection criteria for retrieving a relevant subset of publications. First, we only selected papers published up to May 31st, 2019 that satisfied all of the following three *Inclusion Criteria* (IC):

IC1 The paper is available online and in English;
IC2 The paper should be about *configurable software systems*.
IC3 The paper deals with techniques to statistically learn data from a sample of configurations (see Section 4.1). When different extensions of a paper were observed, *e.g.*, an algorithm is improved by parameter tuning, we intentionally classified and evaluated them as separate primary studies for a more rigorous analysis.

Moreover, we excluded papers that satisfied at least one of the following four *Exclusion Criteria* (EC):

EC1 Introductions to special issues, workshops, tutorials, conferences, conference tracks, panels, poster sessions, as well as editorials and books.
EC2 Short papers (less than or equal to 4 pages) and work-in-progress.
EC3 Pure artificial intelligence papers.
EC4 Secondary studies, such as literature reviews, articles presenting lessons learned, position or philosophical papers, with no technical contribution. However, the references of these studies were read in order to identify other relevant primary studies for inclusion through the snowballing technique (see Section 2.1). Moreover, we consider secondary studies in the related work section (see Section 7).

We find it useful to give some examples of approaches that were *not* included:

- the use of sampling techniques without learning (*e.g.*, the main application is testing or model-checking a software product line). That is, the sample is not used to train a machine learning model but rather for reducing the cost of verifying a family of products. For a review on sampling for product line testing, we refer to Medeiros et al. (2016), Lopez-Herrejon et al. (2015), do Carmo Machado et al. (2014), Lee et al. (2012) and Thüm et al. (2014). We also discuss the complementary between the two lines of work in Section 5.4;
- the use of state-of-the-art recommendations and visualization techniques for configuration guidance (*e.g.*, Pereira et al. (2018) and Murashkin et al. (2013)) and optimization methods based on evolutionary algorithms (*e.g.*, Guo et al. (2011) and Sayyad et al. (2013)) since a sample of configurations' measurements is not considered.
- the use of learning techniques to predict the existence of a software defect or vulnerability based on source code analysis (*e.g.*, in Putri et al. (2017) and Stuckman et al. (2017), where features do not refer to configurable systems, instead features refer to properties or metrics of the source code).

During this step, 1,301 papers were excluded, yielding a total of 41 selected papers for inclusion in the review process. A fourth step of the filtering was performed to select additional relevant papers through the snowballing process. This step considered all included papers, as well as removed secondary studies, which resulted in the inclusion of 28 additional papers. This resulted in the selection of 69 primary papers for data extraction.

The data extraction. The data extraction process was conducted using a structured extraction form in Google Sheets[7] to synthesize all data required for further analysis in such a way that the research questions can be answered. In addition, Google Sheets allow future contributions to be online updated by shareholders. First, all candidate papers were analyzed regarding the selection criteria. The following data were extracted from each retrieved study:

- Date of search, scientific database, and search string.
- Database, authors, title, venue, publication type (*i.e.*, journal, conference, symposium, workshop, or report), publisher, pages, and publication year.
- Inclusion criteria IC1, IC2, and IC3 (yes or no)?
- Exclusion criteria EC1, EC2, and EC3 (yes or no)?
- Selected (yes or no)? If not selected, justification regarding exclusion.

Once the list of primary studies was decided, each selected publication was then read very carefully and the content data for each selected paper was captured and extracted in a second form. The data extraction aimed to summarize the data from the selected primary studies for further analysis of the research questions and for increasing confidence regarding their relevance. All available documentation from studies served as data sources, such as thesis, websites, tool support, as well as the communication with authors (*e.g.*, emails exchanged). The following data were extracted from each selected paper:

- *RQ1*: Scope of the approach. We classified the approach according to the following six categories: *pure prediction*, *interpretability of configurable systems*, *optimization*, *dynamic configuration*, *mining constraints*, and *evolution*.
- *RQ2*: Adopted sampling and learning technique(s).
- *RQ3*: Information about subject systems (*i.e.*, reference, name, domain, number of features, and valid configurations) and the measurement procedure. We collected data about the measured (non-)functional properties and the adopted strategies of measurement.
- *RQ4*: Evaluation metrics and sample designs used by approaches for the purpose of training and validating machine learning models.

The Web supplementary material (Pereira et al., 2019) provides the results of the search procedure from each of these steps.

3. Literature review pattern: Sampling, measuring, learning

Understanding how the system behavior varies across a large number of variants of a configurable system is essential for supporting end-users to choose a desirable product (Apel et al., 2013). It is also useful for developers in charge of maintaining such software systems. In this context, machine learning-based techniques have been widely considered to predict configurations' behavior and assist stakeholders in making informed decisions. Throughout our review effort, we have observed that such approaches follow a 4-stage process: (1) sampling; (2) measuring; (3) learning; and (4) validation. The aim of this section is to introduce the background behind each of these stages in order to answer the research questions in Section 4.

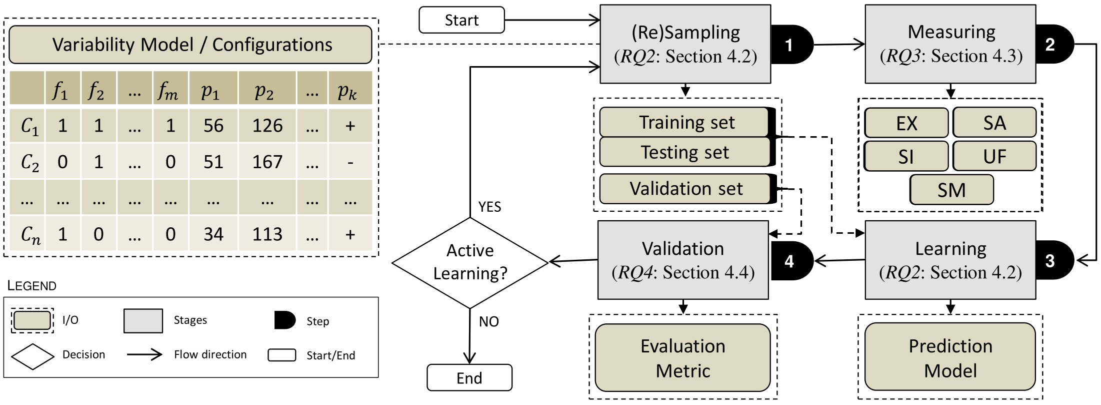
The stages are sketched in Fig. 3. The dashed-line boxes denote the inputs and outputs of each stage. The process starts by building and measuring an initial sample of valid configurations. The set of valid configurations in an SPL is predefined at design time through variability models usually expressed as a feature model (Pohl et al., 2005). Then, these measurements are used to learn a prediction model. Prediction models help stakeholders to better understand the characteristics of complex configurable systems. They try to describe the behavior of all valid configurations. Finally, the validation step computes the accuracy of the prediction model. Some works use active learning (Guo et al., 2017; Oh et al., 2017; Nair et al., 2018; Westermann et al., 2012; Zuluaga et al., 2016; Xi et al., 2012; Grebhahn et al., 2017) to improve the sample in each interaction based on previous accuracy results until it reaches a configuration that has a satisfactory accuracy. Next, we describe in detail each stage.

Sampling. Decision-makers may decide to select or deselect features to customize a system. Each feature can have an effect on a system's *non-functional properties* (NFPs). The quantification of the NFPs of each individual feature is not enough in most cases, as unknown feature interactions among configuration options may cause unpredictable measurements. Interactions occur when combinations among features share a common component or require additional component(s). Thus, understanding the correlation between feature selections and system NFPs is important for stakeholders to be able to find an appropriate system variant that meets their requirements. In Fig. 3, let $C = \{C_1, C_2, \dots, C_n\}$ be the set of n valid configurations, and $C_i = \{f_1, f_2, \dots, f_m\}$ with $f_j \in \{0, 1\}$ a combination of m selected (*i.e.*, 1) and deselected (*i.e.*, 0) features. A straightforward way to determine whether a specific variant meets the requirements is to measure its target NFP P and repeat the process for all C variants of a system, and then *e.g.* search for the cheapest configuration C_i with $C_i \in C$. However, it is usually unfeasible to benchmark all possible variants, due to the exponentially growing configuration space. ML techniques address this issue making use of a small measured *sample* $S_C = \{s_1, \dots, s_k\}$ of configurations, where $S_C \subseteq C$, and the number of samples k and the prediction error ϵ are minimal. With the promise to balance measurement effort and prediction accuracy, several sample strategies have been reported in the literature (see Table A.6). For example, Siegmund et al. (2011, 2012a, 2013b, 2015) explore several ways of sampling configurations, in order to find the most accurate prediction model. Moreover, several authors (Guo et al., 2017; Oh et al., 2017; Nair et al., 2018; Westermann et al., 2012; Zuluaga et al., 2016; Xi et al., 2012; Grebhahn et al., 2017) have tried to improve the prediction power of the model by updating an initial sample based on information gained from the previous set of samples through active learning. The sample might be partitioned into training, testing and validation sets which are used to train and validate the prediction model (see Section 4.4).

Measuring. This stage measures the set of NFPs $\{p_1, \dots, p_l\}$ of a configuration sample $S_C = \{s_1, \dots, s_k\}$, where $p_1 = \{p_1(s_1), \dots, p_1(s_k)\}$. NFPs are measured either by *execution*, *simulation*, *static analysis*, *user feedback* or *synthetic measurements*.

Execution consists of executing the configuration samples and monitoring the measurements of NFPs at runtime. Although execution is much more precise, it may incur unacceptable measurement costs since it is often not possible to create suddenly potentially important scenarios in the real environment. To overcome this issue, some approaches have adopted measurement by simulation. Instead of measuring out of real executions of a system which may result in high costs or risks of failure, *simulation* learns the model using offline environmental conditions that approximate the behavior of the real system faster and cheaper. The use of simulators allows stakeholders to understand the system behavior during early development stages and identify alternative solutions in critical cases. Moreover, simulators can

[7] https://www.google.com/sheets/about/.



Variability Model / Configurations

	f_1	f_2	...	f_m	p_1	p_2	...	p_k
C_1	1	1	...	1	56	126	...	+
C_2	0	1	...	0	51	167	...	-
...	...	...	...	...	...	...	...	...
C_n	1	0	...	0	34	113	...	+

Fig. 3. Employed ML stages to explore SPL configuration spaces.

be programmed offline which eliminates any downtime in online environments. In addition to execution and simulation, *static analysis* infers measurements only by examining the code, model, or documentation. For example, the NFP cost can be measured as the required effort to add a feature to a system under construction by analyzing the system cycle evolution, such as the number of lines of code, the development time, or other functional size metrics. Although static analysis may not be always accurate, it is much faster than collecting data dynamically by execution and simulation. Moreover, partial configurations can be also measured. Finally, instead of measuring configurations statically or dynamically, some authors also make use of either *user feedback* (UF) or *synthetic measurements* (SM). In contrast to static and dynamic measurements, both approaches do not rely on systems artifacts. *User feedback* relies only on domain expert knowledge to label configurations (e.g., whether the configuration is acceptable or not). *Synthetic measurements* are based on the use of learning techniques to generate artificial (non-)functional values to configurable systems (Siegmund et al., 2017). Researchers can use the THOR generator (Siegmund et al., 2017) to mimic and experiment with properties of real-world configurable systems (e.g., performance distributions, feature interactions).

Learning. The aim of this stage is to learn a prediction model based on a given sample of measured configurations $P(S_C)$ to infer the behavior of non-measured configurations $P(C - S_C)$. The sampling set S_C is divided into a *training set* S_T and a *validation set* S_V, where $S_C = S_T + S_V$. The training set is used as input to learn a prediction model, *i.e.* describe how configuration options and their interactions influence the behavior of a system. For parameter tuning, interactive sampling, and active learning, the training set is also partitioned into training and testing sets.

Some authors (Van Aken et al. (2017a), Jamshidi et al. (2017b,a, 2018), Valov et al. (2017)) applied transfer learning techniques to accelerate the learning process. Instead of building the prediction model from scratch, transfer learning reuses the knowledge gathered from samples of other relevant related sources to a target source. It uses a regression model that automatically captures the correlation between target and related systems. Correlation means the common knowledge that is shared implicitly between the systems. This correlation is an indicator that there is a potential to learn across the systems. If the correlation is strong, the transfer learning method can lead to an accurate and reliable prediction model more quickly by reusing measurements from other sources.

Validating. The validation stage quantitatively analysis the quality of the sample S_T for prediction using an evaluation metric on the validation set S_V. To be practically useful, an ideal sample S_T should result in a *(i)* low prediction error; *(ii)* small model size; *(iii)* reasonable measurement effort. The aim is to find as few samples as possible to yield an understandable and accurate ML model in short computation time. In Section 4.4, we detail the evaluation metrics used by the selected primary studies to compute accuracy.

Overall, exploring the configuration space based on a small sample of configurations is a critical step since in practice, the sample may not contain important feature interactions nor reflect the system real behavior accurately. To overcome this issue, numerous learning approaches have been proposed in the last years. Next, we analyze the existing literature by investigating the more fine-grained characteristics of these four learning stages.

4. Results and discussion of the research questions

In this section, we discuss the answers to our research questions defined in Section 1. In Section 4.1, we identify the main goal of the learning process. Next, in Sections 4.2–4.4 we analyze in detail how each study address each learning stage defined in Section 3 to accomplish the goals described in Section 4.1.

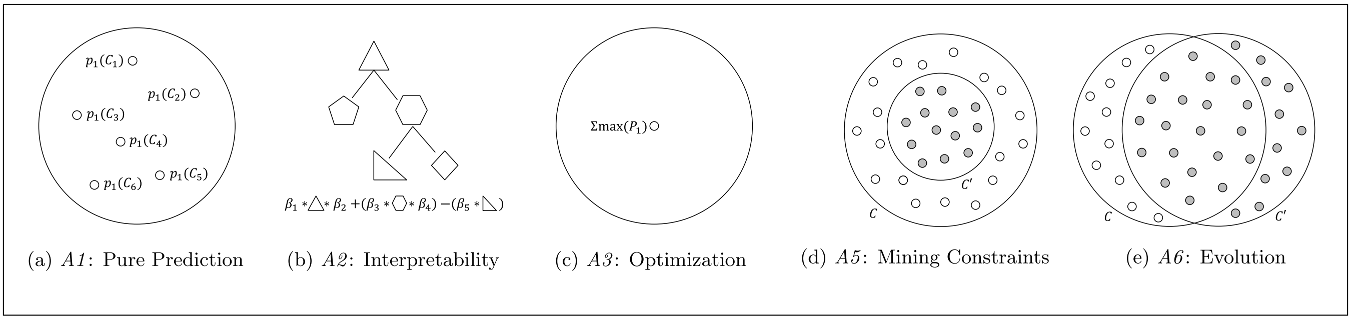
4.1. RQ1: What are the concrete applications of learning software configuration spaces?

In this section, we analyze the application objective of the selected studies since learning may have different practical interests and motivations. Learning techniques have been used in the literature to target six different scenarios (see Fig. 4 and Table 2). In Fig. 4, we represent configurations with small circles and variables of a math formula (i.e. configuration options) with geometric figures. Next, we describe each individual application and give prominent examples of papers entering in this category.

A1 *Pure Prediction.* The aim is to accurately predict labels (i.e., NFPs) p_1 of unmeasured configurations $\{C_1, C_2, \ldots, C_6\}$. Labels can be qualitative (*e.g.*, whether the software configuration has a defect) or quantitative (*e.g.*, the execution time in seconds of a software configuration). The outcome is to associate through prediction some properties to all configurations of the space (see Fig. 4(a)). Guo et al. (2013) is a seminal paper with respect to the use of statistical learning for predicting performances. In this scenario, other factors such as model interpretability and computation cost are less important. In some engineering contexts, the sole prediction of a property of a configuration has limited practical interest *per se* and is sometimes used as a basis for targeting other applications (*e.g.*, configuration optimization).

Table 2
Applicability of selected primary studies. *A1*: Pure prediction; *A2*: Interpretability of configurable systems; *A3*: Optimization; *A4*: Dynamic configuration; *A5*: Mining constraints; *A6*: SPL Evolution.

App.	Reference
A1	Chen et al. (2005), Guo et al. (2013, 2017), Sarkar et al. (2015), Siegmund et al. (2011, 2012a, 2013b, 2015), Valov et al. (2015), Zhang et al. (2015), Yilmaz et al. (2014), Kolesnikov et al. (2017), Couto et al. (2017), Kaltenecker et al. (2019), Jamshidi et al. (2018), Siegmund et al. (2017), Lillack et al. (2013), Siegmund et al. (2013a), Queiroz et al. (2016), Zhang et al. (2016), Song et al. (2013)
A2	Jamshidi et al. (2017a), Kolesnikov et al. (2018), Sincero et al. (2010), Valov et al. (2017), Siegmund et al. (2017), Duarte et al. (2018), Etxeberria et al. (2014)
A3	Oh et al. (2017), Murwantara et al. (2014), Nair et al. (2017, 2018), Siegmund et al. (2012b), Westermann et al. (2012), Martinez et al. (2018), Nair et al. (2018), Van Aken et al. (2017a), Jamshidi and Casale (2016), Zuluaga et al. (2016), Xi et al. (2012), Alipourfard et al. (2017), Zheng et al. (2007), Siegmund et al. (2017, 2008), Ghamizi et al. (2019), Grebhahn et al. (2017), Saleem et al. (2015), Bao et al. (2018), Švogor et al. (2019), El Afia and Sarhani (2017), Ding et al. (2015), Thornton et al. (2013), Xu et al. (2008), Hutter et al. (2011), Osogami and Kato (2007)
A4	Jamshidi et al. (2017b), Weckesser et al. (2018), Samreen et al. (2016), Porter et al. (2007), Jamshidi et al. (2019), Temple et al. (2017a), Krismayer et al. (2017), Sharifloo et al. (2016), Siegmund et al. (2017), Duarte et al. (2018), Chen et al. (2009)
A5	Temple et al. (2016), Acher et al. (2018), Temple et al. (2018), Yilmaz et al. (2006), Gargantini et al. (2017), Amand et al. (2019), Siegmund et al. (2017), Temple et al. (2017a), Krismayer et al. (2017), Safdar et al. (2017)
A6	Sharifloo et al. (2016), Zheng et al. (2007), Siegmund et al. (2017)



(a) *A1*: Pure Prediction (b) *A2*: Interpretability (c) *A3*: Optimization (d) *A5*: Mining Constraints (e) *A6*: Evolution

A4: Dynamic Configuration

Fig. 4. Study application objective.

- A2 *Interpretability of configurable systems.* Understanding the correlation between configuration options and system quality is important for a wide variety of tasks, such as optimization, program comprehension and debugging. To this end, these studies aim at learning an accurate model that is fast to compute and simple to interpret (see the math formula in Fig. 4(b)). For example, Kolesnikov et al. (2018) explore how so-called performance-influence models quantify options' influences and can be used to explain the performance behavior of a configurable system as a whole.
- A3 *Optimization.* Instead of labeling all configurations, optimization approaches aim at finding a (near-)optimal valid configuration to best satisfy requirements, *e.g.*, $\sum max(P_1)$ (see Fig. 4(c)). According to Ochoa et al. (2018), there are three types of stakeholders' requirements: resource constraints (threshold such as `response time` < `1 hour`), stakeholders' preferences (*e.g.*, `security` is extremely more preferable and relevant than `response time`) and optimization objectives (*e.g.*, minimization of `response time`). Although the specification of requirements may reduce the configuration space (Acher et al., 2013; Bak et al., 2016; Ochoa et al., 2015; Eichelberger et al., 2016; Roos-Frantz et al., 2012), searching for the most appropriate configuration is still an overwhelming task due to the combinatorial explosion. Learning approaches exposed in *e.g.* Oh et al. (2017) and Nair et al. (2017, 2018) propose a recursive search method to interactively add new samples to train the prediction model until it reaches a (near-)optimal configuation with an acceptable accuracy.
- A4 *Dynamic Configuration.* There are many dynamic systems (*e.g.*, robotic systems) that aim to manage run-time adaptations of software to react to (uncertain) environmental changes. Without self-adaptation, requirements would be violated. There are several works that explicitly define a set of adaptation rules in the variability model during design-time. Adaptation rules explicitly define under which circumstances a reconfiguration should take place (Pereira et al., 2018). However, anticipating all contextual changes and defining appropriate adaptation rules earlier is often hard for domain engineers due to the huge variability space and the uncertainty of how the context may change at run-time. Therefore, approaches classified in this group use learning techniques to constantly monitor the environment to detect contextual changes that require the system to adapt at runtime. It learns the influences of contexts online in a feedback loop under time and resource constraints through at least one of the application objectives: (*A*1) Pure Prediction, (*A*2) Interpretability, (*A*3) Optimization, (*A*5) Mining Constraints, (*A*6) Evolution (see Fig. 4). For example, learning techniques are used in the dynamic scenario to support the synthesis of contextual-variability models, including logical constraints (Temple et al., 2017a; Krismayer et al., 2017). Other approaches (Jamshidi et al., 2017b; Weckesser et al., 2018; Samreen et al., 2016; Porter et al., 2007; Jamshidi et al., 2019) use learning techniques with the goal of finding a configuration that is optimal and consistent with the current, dynamic context (*e.g.*, given a specific budget).
- A5 *Mining Constraints.* In a configurable system, not all combinations of options' values are possible (*e.g.*, some options are mutually exclusive). Variability models are used to precisely define the space of valid configurations, typically through the specification of logical constraints among options. However, the identification of constraints is a difficult task and it is easy to forget a constraint leading to configurations that do not compile, crash at run-time, or do not meet a particular resource constraint or optimization goal (Temple

et al., 2018). To overcome this issue, learning techniques can be used to discover additional constraints that would exclude unacceptable configurations. These studies seek an accurate and complete set of constraints to restrict the space of possible configurations (see the restricted configurations into the small circle in Fig. 4(d)). Finally, it creates a new variability model by adding the identified constraints to the original model. Therefore, the aim is to accurately remove invalid configurations that were never derived and tested before. Mining constraints approaches work mainly with qualitative properties, such as *video quality* (Temple et al., 2016, 2018) and *defects* (Yilmaz et al., 2006; Krismayer et al., 2017; Gargantini et al., 2017; Amand et al., 2019).

A6 *Evolution.* In the evolution scenario, a configurable system will inevitably need to adapt to satisfy real-world changes in external conditions, such as changes in requirements, design and performance improvements, and changes in the source code. Thus, new configuration options become available and valid, while existing configurations may become obsolete and invalid (see an example of obsolete configurations in Fig. 4(e) represented by unfilled circles). Consequently, the new variability model structure may influence certain NFPs. In this scenario, it is important to make sure that the learning stage is informed by *evolution* about changes and the set of sample configurations is readjusted accordingly with the new variability model by excluding invalid configurations ($C - C'$) and considering the parts of the configuration space not yet covered ($C' - C$). In this context, there are a few works (e.g., Jamshidi et al. (2017a, 2018)) that use transfer learning techniques accross software versions.

> The use of machine learning techniques to learn software configuration spaces has a wide application objective and can be used for supporting developers or end-users in six main different tasks: *Pure Prediction*, *Interpretability of Configurable Systems*, *Optimization*, *Dynamic Configuration*, *Mining Constraints*, and *SPL Evolution*. It is also possible to combine different tasks (*e.g.*, mining constraints for supporting dynamic configuration (Temple et al., 2017a; Krismayer et al., 2017)). There is still room to target other applications (*e.g.*, learning of multiple and composed configurable systems).

4.2. RQ2: Which sampling methods and learning techniques are adopted when learning software configuration spaces?

In this section, we analyze the set of sampling methods used by learning-based techniques in the literature. Also, we aim at understanding which learning techniques were applied and in which context. The complete list of sampling (resp. learning) references can be found in Appendix A (resp. Appendix B). The interested reader can also visualize the relationship between both in our companion Website (Pereira et al., 2019).

Random sampling. Several studies have used random sampling (Guo et al., 2013, 2017; Jamshidi et al., 2017b,a; Oh et al., 2017; Nair et al., 2018; Temple et al., 2016, 2017a; Valov et al., 2015; Weckesser et al., 2018; Zhang et al., 2015; Acher et al., 2018; Jamshidi et al., 2019; Temple et al., 2018; Siegmund et al., 2015; Valov et al., 2017; Nair et al., 2017, 2018; Kaltenecker et al., 2019) with different notions of randomness.

Guo et al. (2013) consider four sizes of random samples for training: N, $2N$, $3N$, and M, where N is the number of features of a system, and M is the number of minimal valid configurations covering each pair of features. They choose size N, $2N$, and $3N$, because measuring a sample whose size is linear in the number of features is likely feasible and reasonable in practice, given the high cost of measurements by execution (see Section 4.3). Guo et al. (2017) and Valov et al. (2015, 2017) use a random sample to train, but also to cross-validate their machine learning model. Several works (Nair et al., 2018, 2017; Oh et al., 2017; Zhang et al., 2015) seek to determine the number of samples in an adaptive, progressive sampling manner and a random strategy is usually employed. Nair et al. (2018, 2017) and Oh et al. (2017) aim at optimizing a configuration. At each iteration, they randomly add an arbitrary number of configurations to learn a prediction model until they reach a model that exhibits a desired satisfactory *accuracy*. They consider several sizes of samples from tens to thousands of configurations. To focus on a reduced part of the configuration space, Nair et al. (2018) and Oh et al. (2017) determine statistically significant parts of the configuration space that contribute to good performance through active learning. In order to have a more representative sample, Valov et al. (2017) adopted stratified random sampling. This sampling strategy exhaustively divides a sampled population into mutually exclusive subsets of observations before performing actual sampling.

Prior works (Guo et al., 2013, 2017; Valov et al., 2017) relied on the random selection of features to create a configuration, followed by a filter to eliminate invalid configurations (*a.k.a*, pseudo-random sampling). Walker's alias sampling (Valov et al., 2017) is an example of pseudo-random sampling. Quasi-random sampling (*e.g.*, sobol sampling) is similar to pseudo-random sampling, however they are specifically designed to cover a sampled population more uniformly (Alipourfard et al., 2017; Valov et al., 2017). However, pseudo-random sampling may result in too many invalid configurations, which makes this strategy inefficient. To overcome this issue, several works (Nair et al., 2018; Oh et al., 2017; Acher et al., 2018; Zhang et al., 2015; Temple et al., 2016, 2017a, 2018; Weckesser et al., 2018; Jamshidi et al., 2017b,a, 2019) use solver-based sampling techniques (*a.k.a.*, true random sampling).

Sampling and heuristics. Instead of randomly choosing configurations as part of the sample, several heuristics have been developed. The general motivation is to better cover features and features' interactions as part of the sample. The hope is to *better capture the essence of the configuration space with a lower sampling size*. We describe some heuristics hereafter.

Knowledge-wise heuristic. This heuristic selects a sample of configurations based on its influence on the target NFPs. The sampling method described by Siegmund et al. (2011, 2013b) measures each feature in the feature model plus all known feature interactions defined by a domain expert. Experts detect feature interactions by analyzing the specification of features, implementation assets, and source code, which require substantial domain knowledge and exhaustive analysis. SPLCoqueror[8] provides to stakeholders an environment in which they can document and incorporate known feature interactions. For each defined feature interaction, a single configuration is added to the set of samples for measurement. THOR (Siegmund et al., 2017) is a generator for synthesizing synthetic yet realistic variability models where users (researchers) can specify the number of interactions and the degree of interactions. Still, these works call to investigate further questions, such as how to better synthesize knowledge actionable by software developers.

Feature-coverage heuristic. To automatically detect all first-order feature interactions, Siegmund et al. (2011, 2012a, 2013b, 2015) use a pair-wise measurement heuristic. This heuristic assumes the existence of a feature interaction between each pair of features in an SPL. It includes a minimal valid configuration for

[8] http://fosd.de/SPLConqueror.

each pair of features being selected. Pair-wise requires a number of measurements that is quadratic in the number of optional features. Some authors (Sarkar et al., 2015; Kaltenecker et al., 2019; Lillack et al., 2013) also use a 3-wise feature coverage heuristic to discover interactions among 3 features. Siegmund et al. (2012a) propose a *3rd-order coverage heuristic* that considers each minimal valid configuration where three features interact pair-wise among them (adopted by Lillack et al. (2013)). They also propose the idea that there are hot-spot features that represent a performance-critical functionality within a system. These hot-spot features are identified by counting the number of interactions per features from the feature-coverage and higher-order interaction heuristics. Yilmaz et al. (2014) adopted even a 4-wise feature coverage heuristic, and Yilmaz et al. (2006) a 5 and 6-wise heuristic. As there are *n*th order feature coverage heuristics, the sample set might be likely unnecessarily large which increases measurement effort substantially. However, not every generated sample contains features that interact with each other. Thus, the main problem of this strategy is that it requires prior knowledge to select a proper coverage criterion. To overcome this issue, state-of-the-art approaches might use the interaction-wise heuristic to fix the size of the initial sample to the number of features or potential feature interactions of a system (Siegmund et al., 2017).

Feature-frequency heuristic. The feature-frequency heuristic considers a set of valid configurations in which each feature is selected and deselected, at least, once. Sarkar et al. (2015) propose a heuristic that counts the number of times a feature has been selected and deselected. Sampling stops when the count of features that were selected or deselected reaches a predefined threshold. Nair et al. (2017) analyze the number of samples required by using the previous heuristic (Sarkar et al., 2015) against a rank-based random heuristic. Siegmund et al. (2011, 2012a,b, 2013b, 2015) quantify the influence of an individual feature by computing the delta of two minimal configurations with and without the feature. They then relate to each feature a minimum valid configuration that contains the current feature, which requires the measurement of a configuration per feature. Hence, each feature can exploit the previously defined configuration to compute its delta over a performance value of interest. In addition, to maximize the number of possible interactions, Siegmund et al. (2015) also relate to each feature a maximal valid configuration that contains the current feature.

There are several others sampling heuristics, such as *Plackett–Burman design* (Siegmund et al., 2015; Grebhahn et al., 2017) for reasoning with numerical options; *Breakdown* (Westermann et al., 2012) (random breakdown, adaptive random breakdown, adaptive equidistant breakdown) for breaking down (in different sectors) the parameter space; *Constrained-driven sampling* (Gargantini et al., 2017) (constrained CIT, CIT of constraint validity, constraints violating CIT, combinatorial union, unconstrained CIT) to verify the validity of combinatorial interaction testing (CIT) models; and many others (see Table A.6).

Sampling and transfer learning. Jamshidi et al. (2017b,a, 2018) aim at applying transfer learning techniques to learn a prediction model. Jamshidi et al. (2017b) consider a combination of random samples from *target* and *source* systems for training: {0%, 10%, …, 100%} from the total number of valid configurations of a source system, and {1%, 2%, …, 10%} from the total number of valid configurations of a target system. In a similar scenario, Jamshidi et al. (2017a) randomly select an arbitrary number of valid configurations from a system before and after environmental changes (*e.g.*, using different hardware, different workloads, and different versions of the system). In another scenario, Jamshidi et al. (2018) use transfer learning to sample. Their sampling strategy, called L2S, exploits common similarities between source and target systems. L2S progressively learns the interesting regions of the target configuration space, based on transferable knowledge from the source.

Arbitrarily chosen sampling. Chen et al. (2005) and Murwantara et al. (2014) have arbitrarily chosen a set of configurations as their sample is based on their current available resources. Sincero et al. (2010) use a subset of valid configurations from a preliminary set of (de)selected features. This sample is arbitrarily chosen by domain experts based on the use of features which will probably have a high influence on the properties of interest. In the context of investigating temporal variations, Samreen et al. (2016) consider on-demand instances at different times of the day over a period of seven days with a delay of ten minutes between each pair of runs. In a similar context, Duarte et al. (2018) and Chen et al. (2009) also sample configurations under different workloads (e.g., active servers and requests per second) at different times of the day. An important insight is that there are engineering contexts in which the sampling strategy is imposed and can hardly be controlled.

All configurations (no sampling). Sampling is not applicable for four of the selected primary studies (Kolesnikov et al., 2018, 2017; Couto et al., 2017; Zheng et al., 2007), mainly for experimental reasons. For example, Kolesnikov et al. (2018, 2017) consider all valid configurations in their experiments and use an established learning technique to study and analyze the trade-offs among prediction error, model size, and computation time of performance-prediction models. For the purpose of their study, they were specifically interested to explore the evolution of the model properties to see the maximum possible extent of the corresponding trade-offs after each iteration of the learning algorithm. So, they performed a whole-population exploration of the largest possible learning set (*i.e.*, all valid configurations). In a similar scenario, Kolesnikov et al. (2017) explored the use of control-flow feature interactions to identify potentially interacting features based on detected interactions from performance prediction techniques using performance measurements). Therefore, they also performed a whole-population exploration of all valid configurations.

Reasoning about configuration validity. Sampling is realized either out of an enumerated set of configurations (e.g., the whole ground truth) or a variability model (e.g., a feature model). The former usually assumes that configurations of the set are logically valid. The latter is more challenging, since picking a configuration boils down to resolve a satisfiability or constraint problem.

Acher et al. (2018) and Temple et al. (2016, 2017a, 2018) encoded variability models as *Constraint Programming* (CSP) by using the Choco solver, while Weckesser et al. (2018) and Siegmund et al. (2017) employed SAT solvers. Constraint solver may produce clustered configurations with similar features due to the way solvers enumerate solutions (*i.e.*, often the sample set consists of the closest *k* valid configurations). Therefore, these strategies do not guarantee true randomness as in pseudo-random sampling. Moreover, using CSP and SAT solvers to enumerate all valid configurations are often impractical (Mendonca et al., 2008; Pohl et al., 2011). Thus, Oh et al. (2017) encoded variability models as *Binary Decision Diagrams* (BDDs) (Akers, 1978), for which counting the number of valid configurations is straightforward. Given the number of valid configurations *n*, they randomly and uniformly select the *k*th configuration, where $k \in \{1 \ldots n\}$, *a.k.a.* randomized true-random sampling. Kaltenecker et al. (2019) perform a comparison among pseudo-random sampling, true-random sampling, and randomized true-random sampling.

Reasoning with numerical options. Ghamizi et al. (2019) transform numeric and enumerated attributes into alternative Boolean features to be handled as binary options. Temple et al. (2016, 2017a, 2018) adopted random sampling of numeric options, *i.e.*

real and integer values. First, their approach randomly selects a value for each feature within the boundaries of its domain. Then, it propagates the values to other features with a solver to avoid invalid configurations. In a similar scenario, Siegmund et al. (2015) and Grebhahn et al. (2017) adopted pseudo-random sampling of numeric options. Siegmund et al. (2015) claim that it is very unusual that numeric options have value ranges with undefined or invalid holes and that constraints among numeric options appear rarely in configurable systems. Grebhahn et al. (2017) adopted different reasoning techniques over binary and numeric options, then they compute the Cartesian product of the two sets to create single configurations used as input for learning. In the scenario of reasoning with numerical options, Amand et al. (2019) arbitrarily select equidistant parameter values.

Numeric sampling have substantially different value ranges in comparison with binary sampling. The number of options' values to select can be huge while constraints should still be respected. Sampling with numeric options is still an open issue — not a pure SAT problem.

> Though random sampling is a widely used baseline, numerous other sampling algorithms and heuristics have been devised and described. There are a set of trade-offs to be considered when sampling configurations: *(1)* minimization of invalid configurations due to constraints' violations among options; *(2)* minimization of the cost (*e.g.*, size) of the sample; *(3)* generalization of the sample to the whole configuration space. The question of a one-size-fits-all sampling strategy remains open and several factors are to be considered: targeted application, subject systems, functional and NFPs, presence of domain knowledge, etc.

Learning techniques. Supervised learning problems can be grouped into regression and classification problems. In both cases, the goal is to construct a machine learning model that can predict the behavior of a configurable system. The difference between the two problems is the fact that the value to predict is numerical for regression and categorical for classification. In this literature review, we found a similar dichotomy, depending on the targeted use-case and the NFP of interest.

A *regression problem* is when the output is a real or continuous value, such as time or CPU power consumption. Most of the learning techniques tackle a supervised regression problem.

CART for regression. Several authors (Guo et al., 2013, 2017; Nair et al., 2017, 2018; Jamshidi et al., 2017a; Murwantara et al., 2014; Sarkar et al., 2015; Temple et al., 2016, 2017a) use the *Classification And Regression Trees* (CART) technique, to model the correlation between feature selections and performance. CART recursively partitions the sample into smaller clusters until the performance of the configurations in the clusters is similar. These recursive partitions are represented as a binary decision tree. For each cluster, these approaches use the sample mean of the performance measurements (or even the majority vote) as the local prediction model of the cluster. So, when they need to predict the performance of a new configuration not measured so far, they use the decision tree to find the cluster which is most similar to the new configuration. Each split of the set of configurations is driven by the (de)selection of a feature that would minimize a prediction error.

CART uses two parameters to automatically control the recursive partitioning process: *minbucket* and *minsplit*. Minbucket is the minimum sample size for any leaf of the tree structure; and minsplit is the minimum sample size of a cluster before it is considered for partitioning. Guo et al. (2013) compute minbucket and minsplit based on the size of the input sample, *i.e.*, if $|S_C| \leq 100$, then minbucket = $|\frac{|S_C|}{10} + \frac{1}{2}|$ and minsplit = 2× minbucket; if $|S_C| > 100$, then minsplit = $|\frac{|S_C|}{10} + \frac{1}{2}|$ and minbucket = $|\frac{minsplit}{2}|$; the minimum of minbucket is 2; and the minimum of minsplit is 4. It should be noted that CART can also be used for classification problems (see hereafter).

Instead of using a set of empirically-determined rules, Guo et al. (2017) combine the previous CART approach (Guo et al., 2013) with automated resampling and parameter tuning, which they call a data-efficient learning approach (DECART). Using resampling, DECART learns a prediction model by using different sample designs (see Section 4.4). Using parameter tuning, DECART ensures that the prediction model has been learned using optimal parameter settings of CART based on the currently available sample. They compare three parameter-tuning techniques: random search, grid search, and Bayesian optimization. Westermann et al. (2012) also used grid search for tuning CART parameters.

Approaches suggested by Nair et al. (2017, 2018) and Sarkar et al. (2015) build a prediction model in a progressive way by using CART. They start with a small training sample and subsequently add samples to improve performance predictions based on the model accuracy (see Section 4.4). In each step, while training the prediction model, Nair et al. (2017) compare the current accuracy of the model with the previous accuracy from the prior iteration (before adding the new set of configurations to the training set). If the current accuracy (with more data) does not improve the previous accuracy (with less data), then the learning reaches a termination criterion (*i.e.*, adding more samples will not result in significant accuracy improvements).

Performance-influence models. Siegmund et al. (2015) combine machine learning and sampling heuristics to build so-called performance-influence models. A step-wise linear regression algorithm is used to select relevant features as relevant terms of a linear function and learn their coefficient to explain the observations. In each iterative step, the algorithm selects the sample configuration with the strongest influence regarding prediction accuracy (*i.e.*, yields the model's lowest prediction error) until improvements of model accuracy become marginal or a threshold for expected accuracy is reached (below 19%). The algorithm concludes with a backward learning step, in which every relevant feature is tested for whether its removal would decrease model accuracy. This can happen if initially a single feature is selected because it better explains the measurements, but it becomes obsolete by other features (*e.g.*, because of feature interactions) later in the learning process. Linear regression allows them to learn a formula that can be understood by humans. It also makes it easy to incorporate domain knowledge about an option's influence on the formula. However, the complete learning of a model using this technique required from 1 to 5 h, depending on the size of the learning set and the size of the models. Kolesnikov et al. (2018) investigate how significant are the trade-offs among prediction error, model size, and computation time.

Other learning algorithms for regression. Westermann et al. (2012) used Multivariate Adaptive Regression Splines (MARS), Genetic Programming (GP), and Kriging. Zhang et al. (2015) used Fourier learning algorithm. In all these works, for a set of samples, it verifies if the resulting accuracy is acceptable for stakeholders (*e.g.*, prediction error rate below 10%). While the accuracy is not satisfactory, the process continues by obtaining an additional sample of measured configurations and iterates again to produce an improved prediction model. Sarkar et al. (2015) propose a sampling cost metric as a stopping criterion, where the objective is to ensure the most optimal trade-off between measurement effort and prediction accuracy. Sampling stops when the count of features selected and deselected passes a predefined threshold.

Chen et al. (2005) propose a linear regression approach to describe the generic performance behavior of application server components running on component-based middleware technologies. The model focuses on two performance factors: workload and degree of concurrency. Sincero et al. (2010) employ analysis of covariance for identifying factors with significant effects on the response or interactions among features. They aim at proposing a configuration process where the user is informed about the impact of their feature selection on the NFPs of interest.

Murwantara et al. (2014) use a set of five ML techniques (*i.e.*, Linear regression, CART, Multilayer Perceptrons (MLPs), Bagging Ensembles of CART, and Bagging Ensembles of MLPs) to learn how to predict the energy consumption of web service systems. They use WEKA's implementation of these techniques with its default parameters (Hall et al., 2009).

Learning to rank. Instead of predicting the raw performance value, it can be of interest to predict the rank of a configuration (typically to identify *optimal* configurations, see RQ1).

Oh et al. (2017) adopt statistical learning techniques to progressively shrink a configuration space and search for near-optimal configurations. First, the approach samples and measures a set of configurations. For each pair of sampled configurations, this approach identifies features that are common (de)selected. Then, it computes the performance influence of each common decision to find the best regions for future sampling. The performance influence measures the average performance over the samples that have the feature selected against the samples that have the feature deselected, *i.e.*, the sample is partitioned by whether a configuration includes a particular feature or not. In addition, Welch's t-test evaluates whether the performance mean of one sample group is higher than the other group with 95% confidence. The most influential decisions are added to the sample. This process continues recursively until they identify all decisions that are statistically certain to improve program performance. They call this a *Statistical Recursive Searching* technique.

Nair et al. (2017) compute accuracy by using the mean rank difference measurement (the predicted rank order is compared to the optimal rank order). They demonstrate that their approach can find optimal configurations of a software system using fewer measurements than the approach proposed by Sarkar et al. (2015). One drawback of this approach is that it requires a holdout set, against which the current model (built interactively) is compared. To overcome this issue, instead of making comparisons, Nair et al. (2018) consider a predefined stopping criterion (budget associated with the optimization process). While the criterion is not met, the approach finds the configuration with the best accuracy and add the configuration to the training set. Consequently, in each step, the approach discards less satisfactory configurations which have a high probability of being dominated. This process terminates when the predefined stopping condition is reached. Nair et al. (2018) demonstrate that their approach is much more effective for multi-objective configuration optimization than state-of-the-art approaches (Zuluaga et al., 2016; Sarkar et al., 2015; Nair et al., 2017).

Martinez et al. (2018) propose the use of data mining interpolation techniques (*i.e.*, similarity distance, similarity radius, weighted mean) for ranking configurations through user feedback on a configuration sample. They estimate the user perceptions of each feature by computing the value of the chi-squared statistic with respect to the correlation score given by users on configurations.

Transfer learning. The previous approaches assume a static environment (*e.g.*, hardware, workload) and NFP such that learning has to be repeated once the environment and NFP changes. In this scenario, transfer learning techniques are adopted to reuse (already available) knowledge from other relevant sources to learn a performance for a target system instead of relearning a model from scratch (Valov et al., 2017; Jamshidi et al., 2017b,a, 2018).

Valov et al. (2017) investigate the use of transfer learning across different hardware platforms. They used 25 different hardware platforms to understand the similarity of performance prediction. They created a prediction model using CART. With CART, the resulting prediction models can be easily understood by end users. To transfer knowledge from a related source to a target source, they used a simple linear regression model.

Jamshidi et al. (2017b) use *Gaussian Process Models* (GPM) to model the correlation between source and target sources using the measure of *similarity*. GPM offers a framework in which predictions can be done using mean estimates with a confidence interval for each estimation. Moreover, GPM computations are based on linear algebra which is cheap to compute. This is especially useful in the domain of dynamic SPL configuration where learning a prediction model at runtime in a feedback loop under time and resource constraints is typically time-constrained.

Jamshidi et al. (2017a) combine many statistical and ML techniques (*i.e.*, *Pearson linear correlation*, *Kullback–Leibler divergence*, *Spearman correlation coefficient*, *paired t-test*, *CART*, *step-wise linear regression*, and *multinomial logistic regression*) to identify when transfer learning can be applied. They use CART for estimating the relative importance of configuration options by examining how the prediction error will change for the trained trees on the source and target. To investigate whether interactions across environments will be preserved, they use step-wise linear regression models (*a.k.a.*, performance-influence models). This model learns all pairwise interactions, then it compares the coefficients of the pairwise interaction terms independently in the source and target environments. To avoid exploration of invalid configurations and reduce measurement effort, they use a multinomial logistic regression model to predict the probability of a configuration being invalid, then they compute the correlation between the probabilities from both environments.

Jamshidi et al. (2018) propose a sampling strategy, called L2S, that exploits common similarities across environments from Jamshidi et al. (2017a). L2S extracts transferable knowledge from the source to drive the selection of more informative samples in the target environment. Based on identifying interesting regions from the performance model of the source, it generates and selects configurations in the target environment iteratively.

Classification problem. Temple et al. (2016, 2017a) and Acher et al. (2018) use CART to infer constraints to avoid the derivation of invalid (non-acceptable) configurations. CART considers a path in a tree as a set of decisions, where each decision corresponds to the value of a single feature. The approach creates new constraints in the variability model by building the negation of the conjunction of a path to reach a faulty leaf. They learn constraints among Boolean and numerical options. As an academic example, Acher et al. (2018) introduce Vary LaTeX to guide researchers to meet paper constraints by using annotated LaTeX sources. To improve the learning process, Temple et al. (2018) specifically target low confidence areas for sampling. The authors apply the idea of using an adversarial learning technique, called evasion attack, after a classifier is trained with a Support Vector Machine (SVM). In addition, Temple et al. (2017a) support the specialization of configurable systems for a deployment at runtime. In a similar context, Safdar et al. (2017) infer constraints over multi-SPLs (*i.e.*, they take into account cross-SPLs rules).

Input sensitivity. Several works consider the influence of the input data on the resulting prediction model (Lillack et al., 2013; Ding et al., 2015; Thornton et al., 2013; Xu et al., 2008; Hutter et al., 2011; Grebhahn et al., 2017; El Afia and Sarhani, 2017; Ghamizi et al., 2019). Many authors (Ding et al., 2015; Thornton et al., 2013; Xu et al., 2008; Hutter et al., 2011; Grebhahn et al., 2017; El Afia and Sarhani, 2017; Ghamizi et al., 2019) address input sensitivity in algorithm auto-tuning. They use learning techniques to search for the best algorithmic variants and parameter settings to achieve optimal performance for a given input instance. It is well known that the performance of SAT solver and learning methods strongly depends on making the right algorithmic and parameter choices, therefore SATzilla (Xu et al., 2008) and Auto-WEKA (Thornton et al., 2013) search for the best SAT solver and learning technique for a given input instance. Lillack et al. (2013) treat the variability caused by the input data as the variability of the SPL. As it is not feasible to model every possible input data, they cluster the data based on its relevant properties (*e.g.*, 10 kB and 20 MB in an or-group for file inputs of a compression library).

Reinforcement learning. Sharifloo et al. (2016) use a reinforcement learning technique to automatically reconfigure dynamic SPLs to deal with context changes. In their approach, learning continuously observes measured configurations and evaluates their ability to meet the contextual requirements. Unmet requirements are addressed by learning new adaptation rules dynamically, or by modifying and improving the set of existing adaptation rules. This stage takes software evolution into account to address new contexts faced at run-time.

> Numerous statistical learning algorithms are used in the literature to learn software configuration spaces. The most used are standard machine learning techniques, such as polynomial linear regressions, decision trees, or Gaussian process models. The targeted use-case and the engineering context explain the diversity of solutions: either a supervised classification or regression problem is tackled; the requirements in terms of interpretability and accuracy may differ; there is some innovation in the sampling phase to progressively improve the learning. Still the use of others (more powerful) ML techniques such as deep learning, adversarial learning, and even the idea of learning different models (*e.g.*, one for each application objective) that could co-evolve can be further explored in future works. Also, unsupervised learning is another potential candidate to support the exploration of configurable systems. For instance, clustering techniques can be used to group together configurations and reduce the costly measurements of each individual configuration within a cluster. Still, analyzing its feasibility remains unexplored.

4.3. RQ3: Which techniques are used to gather measurements of functional and non-functional properties of configurations?

The measuring step takes as input a sample of configurations; and measures, for each configuration, their NFPs. In this section, we investigate how the measurement procedures are technically realized.

Most proposals consider NFPs, such as elapsed time in seconds. In essence, NFPs consist of a *name*, a *domain*, a *value*, and a *unit* (Benavides et al., 2005). The domain type of an NFP can be either quantitative (*e.g.*, real and integer) or qualitative (*e.g.*, string and Boolean). *Quantitative (QT)* properties are typically represented as a numeric value, thus they can be measured on a metric scale, *e.g.*, the configuration is executed in 13.63 s. *Qualitative (QL)* properties are represented using an ordinal scale, such as low (−) and high (+); *e.g.*, the configuration produces a high `video quality`.

Measurements can be obtained through five different strategies: *execution* (EX), *static analysis* (SA), *simulation* (SI), *user feedback* (UF), and *synthetic measurements* (SM). Fig. 5 shows that automated execution is by far the most used technique to measure configuration properties.

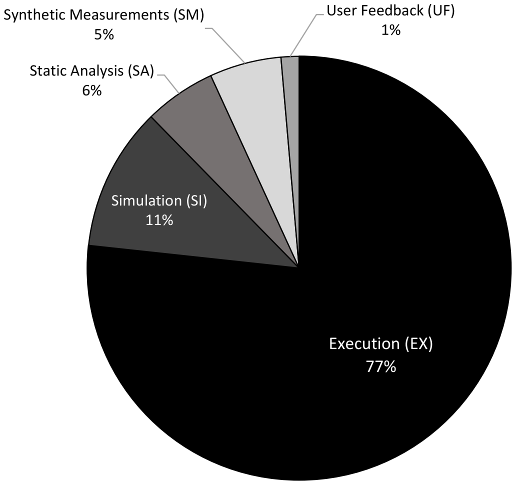


Fig. 5. Strategies applied to measure the sample of configurations.

There are 95 SPLs documented in the literature (see Table C.8 in appendix). They are of different sizes, complexities, implemented in different programming languages (C, C++, and Java), varying implementation techniques (conditional compilation and feature-oriented programming), and from several different application domains (*e.g.*, operating systems, video encoder, database system) and developers (both academic and industrial). Therefore, they cover a broad spectrum of scenarios. It is important to mention that the same subject systems may differ in the number of features, feature interactions, and in the number of valid configurations — the experimental setup is simply different.

Next, we detail the particularities of NFPs. Specifically, we describe how the measurement is performed, what process and strategies are adopted to avoid biases in the results, and also discuss the cost of measuring.

Time. The time spent by a software configuration to realize a task is an important concern and has been intensively considered under different flavors and terminologies (see Fig. 6). Siegmund et al. (2015) invested more than two months (24/7) for measuring the *response time* of all configurations of different subject systems (Dune MGS, HIPAcc, HSMGP, JavaGC, SaC, x264). For each system, they use a different configuration of hardware. The configurations' measurements are reused in many papers, mainly for evaluating the proposed learning process (Guo et al., 2017; Nair et al., 2018; Temple et al., 2017b). Chen et al. (2005) used a simulator to measure the response time of a sequence of 10 service requests by a client to the server. They use two implementation technologies, CORBA and EJB.

Time is a general notion and can be refined according to the application domain. For instance, Kolesnikov et al. (2018) considered *compression time* of a file on lrzip (Long Range ZIP). The measurements were conducted on a dedicated server and repeated multiple times to control measurements noise. Kaltenecker et al. (2019) considered 7-ZIP, a file archiver written in C++, and measured the compression time of the Canterbury corpus.[9]

In another engineering context, several works (Guo et al., 2013; Kolesnikov et al., 2018; Nair et al., 2017; Zhang et al., 2015; Nair et al., 2018; Valov et al., 2015; Kaltenecker et al., 2019) measured the *encoding time* of an input video over 1,152 valid configurations of x264. x264 is a configurable system for encoding video streams into the H.264/MPEG-4 AVC format. As

[9] https://corpus.canterbury.ac.nz/.

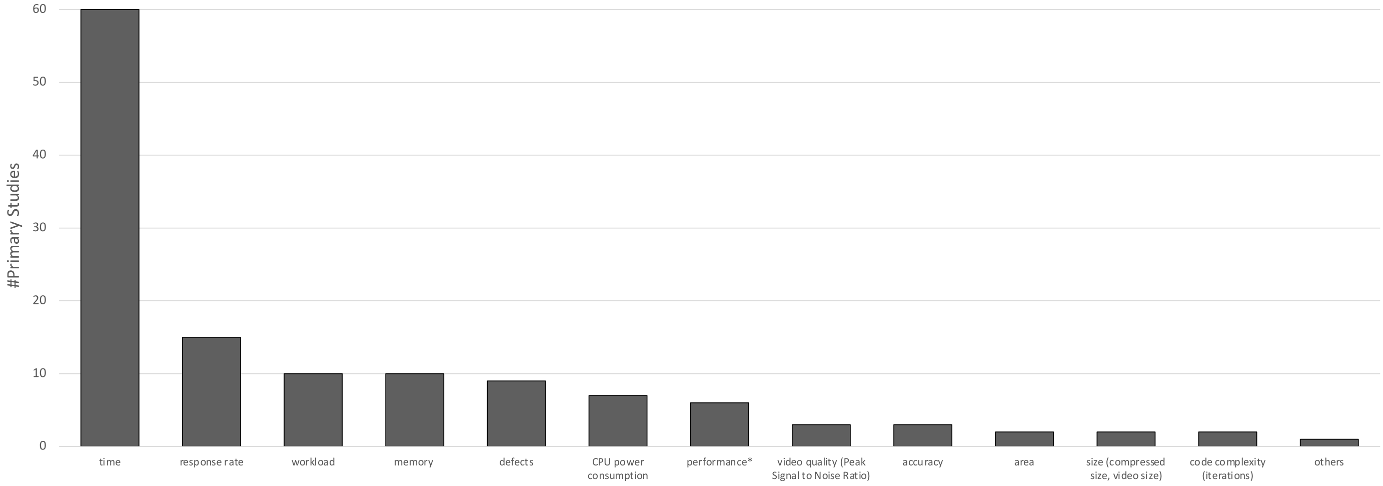


Fig. 6. Non-functional properties measured in the literature; time: response time, encoding time, compression time, I/O time, runtime, speed, timeliness, latency; response rate: throughput, compression ratio; memory: memory footprint, memory consumption, CPU usage.
*Each application domain or even each system has its own specific definition of performance.

benchmark, the Sintel trailer (735 MB) is used and an encoding from AVI is considered. Kaltenecker et al. (2019) measured the encoding time of a short piece of the Big Buck Bunny trailer over 216,000 valid configurations of VPXENC (VP9), a video encoder that uses the VP9 format.

Latency has caught the attention of several researchers. Jamshidi and Casale (2016) and Jamshidi et al. (2017b) measure the average latency (*i.e.*, how fast it can respond to a request) of three stream processing applications on Apache Storm (WordCount, RollingSort, SQL) over a window of 8 min and 2 h, respectively. Jamshidi et al. (2017b) also measure the average latency of the NoSQL database system on Apache Cassandra over a window of 10 min. Jamshidi and Casale (2016) performed the measurements on a multi-node cluster on the EC2 cloud. Jamshidi et al. (2017b) performed the same measurement procedure reported for CPU usage. The measurements used by Nair et al. (2017, 2018) were derived from Jamshidi and Casale (2016). Weckesser et al. (2018) measure the latency of transferred messages over an adaptive Wireless Sensor Networks (WSNs) via simulation. 100 fully-charged nodes were distributed randomly onto a square region for each simulation run. Van Aken et al. (2017a) measure the latency for two OLTP DBMSs (MySQL v5.6, Postgres v9.3) over three workloads (The Yahoo Cloud Serving Benchmark (YCSB), TPC-C, and Wikipedia) during five minutes observation periods. The OLTP DBMS are deployed on m4.large instances with 4 vCPUs and 16 GB RAM on Amazon EC2. They also consider the total execution time of the OLAP DBMS (Actian Vector v4.2) over the TPC-H workload on m3.xlarge instances with 4 vCPUs and 15 GB RAM on Amazon EC2. All of the training data was collected using the DBMSs' default isolation level.

Sharifloo et al. (2016) measure time throughout the evolution of a configurable system (an SPL). In an SPL evolution scenario, the set of measured configurations may include a removed feature or violate changed constraints. In this case, configurations are removed from the sample. Moreover, a modified feature implies to recompute the configuration measurements.

Other NFPs (CPU power consumption, CPU usage, size, etc.) Beyond time, other quantitative properties have been considered.

Nair et al. (2017) measured the compressed size of a specific input over 432 valid configurations of Lrzip, a compression program optimized for large files. Siegmund et al. (2011, 2012b, 2013b) measured the *memory footprint* of nine configurable systems: LinkedList, Prevayler, ZipMe, PKJab, SensorNetwork, Violet, Berkeley DB, SQLite, and Linux kernel. Nair et al. (2018) and Zuluaga et al. (2016) used LLVM, a configurable modular compiler infrastructure. The footprint is measured as the binary size of a compiled configuration.

Several works (Murwantara et al., 2014; Nair et al., 2018; Temple et al., 2017a; Couto et al., 2017; Zuluaga et al., 2016; Jamshidi et al., 2019) measured *CPU power consumption*. Murwantara et al. (2014) used a simulator to compute CPU power consumption over a web service under different loads. They used a kernel-based virtual machine to conduct the measurements based on several combinations of HTTP servers and variant PHP technologies connected to a MySQL database system. They divided the experiment into blocks of 10 s for 100 increasing stages. In the first 10 s, they produce loads of one user per second, and in the next 10 s, they produce the load of two users per second and so on. This results in 1–100 users per period of 10 s. Couto et al. (2017) propose a static learning approach of CPU power consumption. Their approach assures that the source code of every feature from a configuration is analyzed only once. To prove its accuracy, they measured at runtime the worst-case CPU power consumption to execute a given instruction on 7 valid configurations of the disparity SPL (Venkata et al., 2009). They repeated the measurements 200 times for each configuration using the same input. Since their goal was to determine the worst-case CPU power consumption, they removed the outliers (5 highest and lowest values) and retrieved the highest value from every configuration for validation proposes.

Jamshidi et al. (2017b) use simulation measurements. They have repeatedly executed a specific robot mission to navigate along a corridor offline in a simulator and measured performance in terms of *CPU usage* on the CoBot system. To understand the power of transfer learning techniques, they consider several simple hardware changes (*e.g.*, processor capacity) as well as severe changes (*e.g.*, local desktop computer to virtual machines in the cloud).[10] Each measurement took about 30 s. They measured each configuration of each system and environment 3 times.

To overcome the cost of measuring realistic NFPs, Siegmund et al. (2017) proposed a tool, called Thor, to generate artificial and realistic synthetic measurements, based on different distribution patterns of property values for features, interactions, and configurations from a real-world system. Jamshidi et al. (2019) adopted Thor to synthetically measure the property energy consumption

[10] For a complete list of hardware/software variability, see the repository at https://github.com/pooyanjamshidi/transferlearning.

for a robot to complete a mission consisting of randomly chosen tasks within a map.

Qualitative properties. Instead of measuring a numerical value, several papers assign a qualitative value to configurations. In Acher et al. (2018), a variant is considered acceptable or not thanks to an automated procedure. In Temple et al. (2016, 2018), a quantitative value is originally measured and then transformed into a qualitative one through the definition of a threshold. In Queiroz et al. (2016), a variant is considered as acceptable or not based on the static evolution historical analysis of commit messages (*i.e.*, they identify a defect by searching for the following keywords: *bug*, *fix*, *error*, and *fail*). The learning process then aims to predict the class of the configuration — whether configurations are acceptable or not, as defined by the threshold.

Software *defects* are considered in Yilmaz et al. (2006), Krismayer et al. (2017), Gargantini et al. (2017), Yilmaz et al. (2014), Porter et al. (2007), Amand et al. (2019). Yilmaz et al. (2006) and Porter et al. (2007) characterize defects of the ACE+TAO system. Yilmaz et al. (2006) test each supposed valid configuration on the Red Hat Linux 2.4.9-3 platform and on Windows XP Professional using 96 developer-supplied regression tests. In Porter et al. (2007), developers currently run the tests continuously on more than 100 largely uncoordinated workstations and servers at a dozen sites around the world. The platforms vary in versions of UNIX, Windows, Mac OS, as well as to real-time operating systems. To examine the impact of masking effects on Apache v2.3.11-beta and MySQL v5.1, Yilmaz et al. (2014) grouped the test outcomes into three classes: passed, failed, and skipped. They call this approach a *ternary-class fault characterization*. Gargantini et al. (2017) focus on comparing the defect detection capability on different sample heuristics (see Section 4.2), while Amand et al. (2019) deal with comparing the accuracy of several learning algorithms to predict whether a configuration will lead to a defect. In the dynamic configuration scenario, Krismayer et al. (2017) use event logs (via simulation) from a real-world automation SoS to mine different types of constraints according to real-time defects.

Finally, Martinez et al. (2018) consider a 5-point *user likeability* scale with values ranging from 1 (strong dislike) to 5 (strong like). In this work, humans have reviewed and labeled configurations.

Accuracy of measurements. In general, measuring NFPs (*e.g.*, time) is a difficult process since several confounding factors should be controlled. The need to gather measures over numerous configurations exacerbates the problem.

Weckesser et al. (2018) mitigated the construct threat of the inherent randomness and repeated all runs five times with different random seeds. Measurements started after a warm-up time of 5 min. Kaltenecker et al. (2019) measured each configuration between 5 to 10 times until reaching a standard deviation of less than 10%. Zhang et al. (2015) repeated the measurements 10 times.

To investigate the influence of measurement errors on the resulting model, Kolesnikov et al. (2018) and Duarte et al. (2018) injected measurement errors to the original measurements and repeated the learning process with *polluted* datasets. Then, they compared the prediction error of the noisy models to the prediction error of the original models to see the potential influence of measurement errors. For each subject system, Kolesnikov et al. (2018) repeated the learning process five times for different increasing measurement errors.

Dynamic properties are susceptible to measurement errors (due to non-controlled external influences) which may bias the results of the measuring process. To account for measurement noise and be subject to external influences, these properties need to be measured multiple times on dedicated systems. Thus, the total measurement cost to obtain the whole data used in the experiments is overly expensive and time-consuming (*e.g.*, Kaltenecker et al. (2019) spent multiple years of CPU time). According to Lillack et al. (2013), there is a warm-up phase followed by multiple times runs; and the memory must be set up large enough to prevent disturbing effects from the Garbage Collector, as well as all operations, must be executed in memory so that disk or network I/O will also produce no disturbing effects. Most of the works only consider the variability of the subject system, while they use static inputs and hardware/software environments. Therefore, the resulting model may not properly characterize the performance of a different input or environment, since most of the properties (*e.g.*, CPU power consumption and compression time) are dependent of the input task and the used hardware/software. Consequently, hardware/software must also be taken into account as dependent variables as considered by Jamshidi et al. (2017b,a).

There are some properties that are much accurate, because *e.g.* they are not influenced by the used hardware, such as footprint. Siegmund et al. (2013b) parallelized the measurements of the footprint on three systems and used the same compiler. Moreover, the footprint can be measured quickly only once, without measurement bias.

Cost of measurements. The cost of observing and measuring software can be important, especially when multiple configurations should be considered. The cost can be related to computational resources needed (in time and space). It can also be related to human resources involved in labeling some configurations (Martinez et al., 2018).

Zuluaga et al. (2016) and Nair et al. (2018) measure the quantitative NFP *area* of a field-programmable gate array (FPGA) platform consisting of 206 different hardware implementations of a sorting network for 256 inputs. They report that the measurement of each configuration is very costly and can take up to many hours. Porter et al. (2007) characterization of defects for each configuration ranges from 3 h on quad-CPU machines to 12–18 h on less powerful machines. Yilmaz et al. (2006) took over two machine years to run the total of 18,792 valid configurations. In Siegmund et al. (2012b), a single measurement of memory footprint took approximately 5 min. Temple et al. (2016) report that the execution of a configuration to measure video quality took 30 min on average. They used grid computing to distribute the computation and scale the measurement for handling 4,000+ configurations. The average time reported by Murwantara et al. (2014) to measure the CPU power consumption of all sampled configurations was 1,000 s. The authors set up a predefined threshold to speed up the process.

There are thirteen main NFPs supported in the literature (see Fig. 6). Some of them are less costly, such as *code complexity*, which can be measured statically by analyzing the number of code lines (Siegmund et al., 2012b). As a result, the measurements can be parallelized and quickly done only once. Otherwise, dynamic properties, such as *CPU power consumption* and *response time* are directly related to hardware and external influences. While *CPU power consumption* might be measured under different loads over a predefined threshold time, the property *response time* is much costly as a threshold cannot be defined and the user does not know how long it will take. To support the labeling of configurations, some works use synthetic measurements and statistical analysis strategies. Practitioners need to find a sweet spot among the available techniques to have accurate measurements with a small sample.

Depending on the subject system and application domain (see the dataset of Siegmund et al. (2015)), there are more favorable cases with only a few seconds per configuration. However, even in this case, the overall cost can quickly become prohibitive when the number of configurations to measure is too high. In Weckesser et al. (2018), all training simulation runs took approximately 412 h of CPU time.

Numerous qualitative and quantitative properties of configurations are measured mainly through the use of automated software procedures. For a given subject system and its application domain, there may be more than one measure (*e.g.*, CPU power consumption and video quality for x264). Also, the same NFP may be measured in different ways (*e.g.*, video quality can be measured either by *user feedback* or *execution* of a program to automatically attribute labels to features). Time is the most considered performance measure and is obtained in the literature through either execution, simulation, static analysis, or synthetic measurement. The general problem is to find a good tradeoff between the cost and the accuracy of measuring numerous configurations (*e.g.*, simulation can speed up the measurement process at the price of approximating the real observations).

4.4. RQ4: How are learning-based techniques validated?

In this section, we aim at understanding how the validation process is conducted in the literature.

There are five design strategies documented in the literature to explore the sample data for learning and validation (see Table 3).

Merge. Training, testing, and validation sets are merged. Kolesnikov et al. (2018) studied and analyzed the trade-offs among prediction error, model size, and computation time of performance-prediction models. For the purpose of their study, they were specifically interested to explore the evolution of the model properties to see the maximum possible extent of the corresponding trade-offs after each iteration of the learning algorithm. So, they performed a whole-population exploration of the largest possible learning set (*i.e.*, all valid configurations). Therefore, in their validation process, training, testing and validation sets are merged. In a similar scenario, other authors (Chen et al., 2005; Oh et al., 2017; Nair et al., 2018) also used the whole sample for both, learning and validation, even they considered a small set of valid configurations as the sample. Some studies justify the use of a merge pool with the need to compare different sampling methods. However, training the algorithm on the validation pool may introduce bias to the results of the accurateness of the approach. To overcome this issue, studies have used other well-established ML design strategies.

In particular, Guo et al. (2017) study the trade-offs among hold-out, cross-validation, and bootstrapping. For most of the case studies, 10-fold cross-validation outperformed hold-out and bootstrapping. In terms of the running time, although these three strategies usually take seconds to run, Guo et al. (2017) show that hold-out is the fastest one, and 10-fold cross-validation tends to be faster than bootstrapping.

Hold-out (HO). Most of the works (Jamshidi and Casale, 2016; Guo et al., 2013; Jamshidi et al., 2017b; Sarkar et al., 2015; Temple et al., 2016, 2017a; Valov et al., 2015, 2017; Weckesser et al., 2018; Acher et al., 2018; Temple et al., 2018; Nair et al., 2018; Yilmaz et al., 2006; Krismayer et al., 2017; Jamshidi et al., 2018; Zuluaga et al., 2016; Amand et al., 2019; Alipourfard et al., 2017; Nair et al., 2017; Guo et al., 2017; Westermann et al., 2012) used a hold-out design. Hold-out splits an input sample S_C into two disjoined sets S_t and S_v, one for training and the other for validation, *i.e.* $S_C = S_t \cup S_v$ and $S_t \cap S_v = \emptyset$. To avoid bias in the splitting procedure, some works repeated it multiple times. For example, in Valov et al. (2015), the training set for each sample size was selected randomly 10 times. For transfer-learning applications (Jamshidi et al., 2017b; Valov et al., 2017; Jamshidi et al., 2018), the training set comes from samples of the target and related sources, while the validation set comes from samples only from the target source.

Cross-validation (CV). Cross-validation splits an input sample S_C into k disjoined subsets of the same size, *i.e.*, $S = S_1 \cup \cdots \cup S_k$, where $S_i \cap S_j = \emptyset$ $(i \neq j)$; each subset S_i is selected as the validation set S_v, and all the remaining $k-1$ subsets are selected as the training set S_t. As an example, Yilmaz et al. (2014) used a 5-fold cross-validation to create multiple models from different subsets of the input data. In a 5-fold cross-validation, the sample S is partitioned into 5 subsets (*i.e.*, $S = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5$) of the same size. Each subset is selected as the validation set and all of the remaining subsets form the training set. Most authors (Guo et al., 2017; Samreen et al., 2016; Martinez et al., 2018; Murwantara et al., 2014; Safdar et al., 2017) relayed on a 10-fold cross-validation. 10-fold cross-validation follows the same idea of a 5-fold cross-validation, producing $k = 10$ groups of training and validation sets. Guo et al. (2017) show that a 10-fold cross-validation design does not work well for very small samples, *e.g.*, it did not work for an Apache system.

Bootstrapping (BT). Four studies (Van Aken et al., 2017a; Jamshidi and Casale, 2016; Nair et al., 2017; Guo et al., 2017) relayed on the bootstrapping design. Bootstrapping relies on random sampling with replacement. Given an input sample S_C with k configurations, bootstrapping randomly selects a configuration C_b, with $1 \leq b \leq k$ and copies it to the training set S_t. However, it keeps C_b in S_C for the next selection. This process is repeated k times. Given the training sample, bootstrapping uses $S_C \backslash S_t$ as the validation set S_v. Nair et al. (2017) used a non-parametric bootstrap test with 95% confidence for evaluating the statistical significance of their approach. In non-parametric bootstrap, the input sample S_C is drawn from a discrete set, *i.e.*, 95% of the resulting sample S_t should fall within the 95% confidence limits about S_C.

Dynamic sector validation. Westermann et al. (2012) relayed on two types of Dynamic Sector Validation designs: with local prediction error scope (DSL) and with global prediction error scope (DSG). Dynamic Sector Validation decides if a sample configuration $C_i \in S_C$ is part either of the training or testing set based on the sector's prediction error of the adopted learning techniques. In DSL, all sectors have a prediction error that is less than the predefined threshold, while in DSG the average prediction error of all sectors is less than the predefined threshold.

There are studies where a sample design is *not applicable* (Jamshidi et al., 2017a; Gargantini et al., 2017; Kolesnikov et al., 2017; Couto et al., 2017; Porter et al., 2007; Siegmund et al., 2017; Xi et al., 2012; Zheng et al., 2007; Jamshidi et al., 2019; Siegmund et al., 2008; Etxeberria et al., 2014), as well as studies (Sharifloo et al., 2016; Zhang et al., 2015; Siegmund et al., 2013a; Hutter et al., 2011; Osogami and Kato, 2007) without further details about the sample design. Given the sampling design, evaluation metrics are used to (learn and) validate the resulting prediction model. Table 4 sketches which evaluation metrics are used in the literature. The first column identifies the study reference(s). The second and third columns identify the name of the evaluation metric and its application objective, respectively. Similar to Table B.7, here the application objective is related to the scenarios in which each evaluation metric has been already used in the SPL literature. Therefore, in future researches, some metrics may be explored in other scenarios as well. Notice that some studies (Sharifloo et al., 2016; Siegmund et al., 2012b; Sincero et al., 2010) did not report any detail about the validation process.

Evaluation metrics. State-of-the-art techniques rely on 50 evaluation metrics from which it is possible to evaluate the accuracy of the resulting models in different application scenarios. There are metrics dedicated to supervised classification problems (*e.g.*, precision, recall, F-measure). In such settings, the goal is to quantify the ratio of correct classifications to the total number of input samples. For instance, Temple et al. (2016) used precision

Table 3
Sample designs reported in the literature.

Sample design	Reference
Merge	Chen et al. (2005), Oh et al. (2017), Kolesnikov et al. (2018), Nair et al. (2018), Siegmund et al. (2011, 2012a,b, 2013b, 2015), Sincero et al. (2010), Kaltenecker et al. (2019)
Hold-Out	Jamshidi and Casale (2016), Guo et al. (2013), Jamshidi et al. (2017b), Sarkar et al. (2015), Temple et al. (2016, 2017a), Valov et al. (2015, 2017), Weckesser et al. (2018), Acher et al. (2018), Temple et al. (2018), Nair et al. (2018), Yilmaz et al. (2006), Krismayer et al. (2017), Jamshidi et al. (2018), Zuluaga et al. (2016), Amand et al. (2019), Alipourfard et al. (2017), Nair et al. (2017), Guo et al. (2017), Westermann et al. (2012), Queiroz et al. (2016), Zhang et al. (2016), Ghamizi et al. (2019), Lillack et al. (2013), Grebhahn et al. (2017), Saleem et al. (2015), Bao et al. (2018), Švogor et al. (2019), El Afia and Sarhani (2017), Xu et al. (2008), Duarte et al. (2018), Chen et al. (2009), Song et al. (2013)
Cross-Validation	Murwantara et al. (2014), Martinez et al. (2018), Samreen et al. (2016), Yilmaz et al. (2014), Guo et al. (2017), Safdar et al. (2017), Ding et al. (2015), Duarte et al. (2018)
Bootstrapping	Van Aken et al. (2017a), Jamshidi and Casale (2016), Nair et al. (2017), Guo et al. (2017), Thornton et al. (2013)
Dynamic Sector Validation	Westermann et al. (2012)

and recall to control whether acceptable and non-acceptable configurations are correctly classified according to the ground truth. Others (Temple et al., 2018; Porter et al., 2007) use qualitative analysis to identify features with significant effects on defects, and understand feature interactions and decide whether further investigation of features is justified.

There are also well-known metrics for regression problems, such as, *Mean Relative Error* (MRE - Eq. (1)) and *Mean Absolute Error* (MAE - Eq. (2)). These metrics aim at estimating the accuracy between the exact measurements and the predicted one.

$$MRE = \frac{1}{|S_v|} \sum_{C \in S_v} \frac{|C_{(p_i)} - C_{(p'_i)}|}{C_{(p_i)}} \tag{1}$$

$$MAE = \frac{|C_{(p'_i)} - C_{(p_i)}|}{C_{(p_i)}} \tag{2}$$

Where S_v is the validation set, and $C_{(p_i)}$ and $C_{(p'_i)}$ indicate the exact and predicted value of p_i for $C \in S_v$, respectively.

Contributions addressing learning-to-rank problems develop specific metrics capable of assessing the ability of a learning method to correctly rank configurations. For example, Nair et al. (2017, 2018) use the error difference between the ranks of the predicted configurations and the true measured configurations. To get insights about when stop sampling, they also discuss the trade-off between the size of the training set and rank difference.

Interpretability metrics. Some metrics are also used to better *understand* configuration spaces and their distributions, for example, *when* to use a transfer learning technique. In this context, Jamshidi et al. (2017a) use a set of similarity metrics (Kullback–Leibler, Pearson Linear Correlation, Spearman Correlation Coefficient, and Rank Correlation) to investigate the relatedness of the source and target environments. These metrics compute the correlation between predicted and exact values from source and target systems. Moreover, Kolesnikov et al. (2018) and Valov et al. (2017) use size metrics as insights of interpretability. Valov et al. (2017) compare the structure (size) of prediction models built for the same configurable system and trained on different hardware platforms. They measured the size of a performance model by counting the features used in the nodes of a regression tree, while Kolesnikov et al. (2018) define the model size metric as the number of configuration options in every term of the linear regression model. Temple et al. (2016) reported that constraints extracted from decision trees were consistent with the domain knowledge of a video generator and could help developers preventing non-acceptable configurations. In their context, interpretability is important both for validating the insights of the learning and for encoding the knowledge into a variability model.

Sampling cost. Several works (Alipourfard et al., 2017; Sarkar et al., 2015; Weckesser et al., 2018; Nair et al., 2018,?) use a sampling cost measurement to evaluate their prediction approach. In order to reduce measurement effort, these approaches aim at sampling configurations in a smart way by using as few configurations as possible. In Alipourfard et al. (2017), Weckesser et al. (2018) and Nair et al. (2018,?), sampling cost is considered as the total number of measurements required to train the model. These authors show the trade-off between the size of the training set and the accuracy of the model. Nair et al. (2018,?) compare different learning algorithms, along with different sampling techniques to determine exactly those configurations for measurement that reveal key performance characteristics and consequently reach the minimum possible sampling cost. Sarkar et al. (2015) introduce a cost model, which they consider the measurement cost involved in building the training and testing sets, as well as the cost of building the prediction model and computing the prediction error. Notice that sampling cost can be seen as an additional objective of the problem of learning configuration spaces, *i.e.* not only the usual learning accuracy should be considered.

There is a wide range of validation metrics reported in the literature, and some are reused amongst papers (e.g., MRE). However, several metrics can be used for the same task and there is not necessarily a consensus. Most of the studies use a unique metric for validation, which may provide incomplete quantification of the accuracy.

> The evaluation of the learning process requires to confront a trained model with new, unmeasured configurations. Hold-out is the most considered strategy for splitting configuration data into a training set and a testing set. Depending on the targeted task (regression, ranking, classification, understanding), several metrics have been reused or defined. In addition to learning accuracy, sampling cost and interpretability of the learning model can be considered as part of the problem and thus assessed.

5. Emerging research themes and open challenges

In the previous sections, we have given an overview of the state-of-the-art in sampling, measuring, and learning software configuration spaces.[11] Here, we will now discuss open challenges and their implications for research and practice.

There are a few reports of real-world adoption (Kolesnikov et al., 2018; Temple et al., 2016; Jamshidi et al., 2019; Weckesser et al., 2018), but overall it seems that despite wide applicability in terms of application domains, subject systems, or measurement

[11] We consolidate the results from all research questions through a set of bubble charts in our supplementary material at https://github.com/VaryVary/ML-configurable-SLR.

Table 4
Validation procedure reported in the literature. *A1*: Pure prediction; *A2*: Interpretability of configurable systems; *A3*: Optimization; *A4*: Dynamic configuration; *A5*: Mining constraints; *A6*: SPL evolution.

Reference	Evaluation metric	Applicability
Valov et al. (2015)	Closeness Range, Winner Probability	*A1*
Zhang et al. (2016), Song et al. (2013)	Coverage	*A1*
Kolesnikov et al. (2017)	Jaccard Similarity	*A1*
Zhang et al. (2016)	Performance-Relevant Feature Interactions Detection Accuracy	*A1*
Yilmaz et al. (2014)	t-masked metric	*A1*
Queiroz et al. (2016)	True Positive (TP) Rate, False Positive (FP) Rate, Receiver Operating Characteristic (ROC)	*A1*
Chen et al. (2005), Guo et al. (2013, 2017), Kolesnikov et al. (2018), Siegmund et al. (2011, 2012a, 2013b, 2015), Valov et al. (2015, 2017), Westermann et al. (2012), Zhang et al. (2015), Nair et al. (2018), Couto et al. (2017), Kaltenecker et al. (2019), Jamshidi et al. (2019, 2018), Grebhahn et al. (2017), Thornton et al. (2013), Duarte et al. (2018)	Mean Relative Error (MRE)	*A1*, *A2*, *A3*, *A4*
Alipourfard et al. (2017), Sarkar et al. (2015), Weckesser et al. (2018), Nair et al. (2018,?), Grebhahn et al. (2017), Duarte et al. (2018), Song et al. (2013)	Sampling Cost	*A1*, *A2*, *A3*, *A4*
Westermann et al. (2012), Zhang et al. (2015)	Highest Error (HE)	*A1*, *A3*
Jamshidi et al. (2017b), Oh et al. (2017), Murwantara et al. (2014), Martinez et al. (2018), Jamshidi et al. (2018), Jamshidi and Casale (2016), Siegmund et al. (2013a), Ghamizi et al. (2019), Lillack et al. (2013), El Afia and Sarhani (2017), Bao et al. (2018)	Mean Absolute Error (MAE)	*A1*, *A3*, *A4*
Kaltenecker et al. (2019), Safdar et al. (2017), Hutter et al. (2011)	Mann–Whitney U-test	*A1*, *A3*, *A5*
Samreen et al. (2016), Kaltenecker et al. (2019)	F-test	*A1*, *A4*
Temple et al. (2016, 2017a), Acher et al. (2018), Kolesnikov et al. (2017), Amand et al. (2019)	Precision, Recall	*A1*, *A4*, *A5*
Yilmaz et al. (2006, 2014), Amand et al. (2019), Safdar et al. (2017), Queiroz et al. (2016)	Precision, Recall, F-measure	*A1*, *A5*
Etxeberria et al. (2014)	GAP	*A2*
Jamshidi et al. (2017a)	Kullback–Leibler (KL), Pearson Linear Correlation, Spearman Correlation Coefficient	*A2*
Valov et al. (2017), Kolesnikov et al. (2018)	Structure of Prediction Models	*A2*
Jamshidi et al. (2017a, 2019)	Rank Correlation	*A2*, *A4*
Van Aken et al. (2017a), Grebhahn et al. (2017)	Domain Experts Feedback	*A3*
Osogami and Kato (2007)	Error Probability	*A3*
Martinez et al. (2018)	Global Confidence, Neighbors Density Confidence, Neighbors Similarity Confidence	*A3*
Westermann et al. (2012)	LT15, LT30	*A3*
Nair et al. (2017, 2018)	Mean Rank Difference	*A3*
Murwantara et al. (2014)	Median Magnitude of the Relative Error (MdMRE)	*A3*
Zuluaga et al. (2016)	Pareto Prediction Error	*A3*
Bao et al. (2018), Švogor et al. (2019), Saleem et al. (2015)	Rank Accuracy (RA)	*A3*
Van Aken et al. (2017a), Ding et al. (2015), Xu et al. (2008), Osogami and Kato (2007)	Tuning Time	*A3*
Murwantara et al. (2014), El Afia and Sarhani (2017), Samreen et al. (2016)	Mean Square Error (MSE)	*A3*, *A4*
Samreen et al. (2016)	*p*-value, R2, Residual Standard Error (RSE)	*A4*
Chen et al. (2009)	Reward	*A4*
Porter et al. (2007)	Statistical Significance	*A4*
Temple et al. (2018), Porter et al. (2007)	Qualitative Analysis	*A4*, *A5*
Krismayer et al. (2017)	Ranking Constraints	*A4*, *A5*
Safdar et al. (2017)	Delaney's Statistics	*A5*
Safdar et al. (2017)	Distance Function, Hyper-volume (HV)	*A5*
Gargantini et al. (2017)	Equivalence among Combinatorial Models	*A5*
Gargantini et al. (2017)	Failure Index Delta (FID), Totally Repaired Models (TRM)	*A5*

properties, there is little concrete evidence that the proposed learning techniques are actually adopted in widespread everyday practice. For instance, the x264 video encoder is widely considered in the academic literature, but we are not aware of reports that learning-based solutions, as discussed here, have an impact on the way developers maintain it or on the way users configure it.

Hence, in this section, we discuss some scientific and technical obstacles that contribute to this *gap* between what is available in academic literature and what is actually adopted in practice. In doing so, we aim to identify (1) points that should be considered when choosing and applying particular techniques in practice and (2) opportunities for further research. We summarize our findings in Table 5.

5.1. Generalization and transfer learning

A first and important risk when adopting a learning-based approach is that the model may not generalize to new, previously unseen configuration data. Some works have already observed

Table 5
Summary of open challenges: (i) resulting considerations for practical applications and (ii) necessary research.

Open challenge	Considerations in practical applications	Research needed
Generalization and transfer learning (Section 5.1)	Given a new "context" (*e.g.,* hardware change), can a learning model be reused?	Transfer learning from a known configuration space to a new configuration one; identification of factors that influence performance distribution
Interpretability (Section 5.2)	Consider techniques with interpretable representation (*e.g.,* CART, step-wise linear regression); application objectives might determine relative importance of interpretability	Methods for assessing interpretability; techniques that generate an explanation in hindsight
Learning trade-offs (Section 5.3)	Beyond raw ML performance (*e.g.,* accuracy) consider bigger picture (*e.g.,* interpretability, integration in surrounding software system, robustness against changes)	Lack of actionable frameworks to choose techniques based on consideration of multiple goals and trade-offs between them
Sampling (Section 5.4)	No clear relation between application objective and chosen sampling techniques; different strategies perform best on different workloads and NFPs (see *e.g.,* Pereira et al. (2020), Grebhahn et al. (2019))	More work needed on effectiveness and strengths of sampling techniques in different contexts
Cost-effectiveness (Section 5.5)	The cost for executing and measuring configurations can be a barrier	Simulations; distributed infrastructures; handling of uncertainty; transfer learning to reuse knowledge
Evaluation/ application in realistic settings (Section 5.6)	Expect that techniques described in academic literature do not "just work" out of the box	Existing works often evaluated in controlled environments, more work needed where techniques are evaluated/applied in realistic settings
Integrated tools (Section 5.7)	Consider tools that are open for integration in real-world projects	Lack of integrated tools for supporting developers and users

that many factors can influence the performance distribution of a configurable system: the version of the software, the hardware on which the software is executed, the workload, etc. The consequence is that, *e.g.,* a prediction model of a configurable system may reach high accuracy in a given situation (close to the one used for training) and then the performance drops when some of these factors change. The reuse of learning-based models is thus an important concern; the worst-case scenario is to learn a new model for every usage of a configurable system. We notice that several recent works aim to transfer the learning-based knowledge of a configuration space to another setting (*e.g.,* a different hardware setup) (Valov et al., 2017; Jamshidi et al., 2017b,a, 2018). Another research direction is to characterize which and to what extent factors influence the performance distribution of a configurable system. The hope is to propose learning-based solutions that are effective independent of the context in which the configurable software is deployed.

5.2. Interpretability

In many engineering scenarios, developers or users of configurable systems want to understand learning-based models and not treat them as black-boxes. This observation partly explains why decision trees (CART) and step-wise linear regressions are the most used learning algorithms (see Table B.7): the resulting models provide interpretable information, through rules or coefficients. The challenge is to find a good-enough compromise between accuracy and interpretability. Interpretability can be the major objective to fulfill (application A2) or an important concern, *e.g.,* mining constraints (application A5) requires the use of rule-based learning. For other tasks such as optimization (A3), accuracy seems more important and hence the choice of a well-performing learning algorithm might be more important than interpretability.

We notice that there are few studies reported in the literature whose evaluation objective is the comprehension of configurable systems, for instance, to understand which options are influential. However, we are not aware of empirical user studies to validate the learning approach with regards to the comprehension by domain experts. The existing works typically rely on a model size metric to report the level of comprehension (Valov et al., 2017; Kolesnikov et al., 2018). How to evaluate interpretability is an open issue in machine learning (Molnar, 2019) that also applies to the learning of software configuration space.

5.3. Trade-offs when choosing learning algorithms

Another concern to consider is the computation time needed to apply a statistical learning method. Many factors should be considered: the algorithm itself, the size of the training set, the number of features, the number of trees (for random forest), the hyperparameters, etc. Looking at experiments and subject systems considered in the literature, the size of the training set as well as the number of features remain affordable so far: only a few seconds and minutes are needed to train. However, there are some exceptions with studies involving large systems (*e.g.,* Linux see Acher et al. (2019a,b)) or learning procedures that can take hours.

Overall, the choice of a learning algorithm is a tradeoff between different concerns (interpretability, accuracy, training time). Which one practitioners should use and under which conditions? There is still a lack of actionable frameworks that would automatically choose or recommend a suited learning technique based on an engineering context (*e.g.,* targeted NFPs, need of interpretability). So far, tree-based methods constitute a good compromise and can be applied for many applications. More (empirical) research is needed to help practitioners systematically choosing an adequate solution.

5.4. Sampling: the quest for the right strategy

Currently, there is no dominant sampling technique that can be used without further consideration in any circumstance by practitioners. We reported 23 high-level sampling techniques used by state-of-the-art approaches. Data of our systematic review shows that there is no clear relation between the choice of a sampling strategy and a specific application objective (A1, …, A6). Hence, more research is needed to evaluate the effectiveness of particular sampling techniques.

A first specific challenge when sampling over a configurable system is to generate configurations conforming to *constraints*. This boils down to solving a satisfiability problem (SAT) or constraint satisfaction problem (CSP) when numerical values should be handled (Temple et al., 2016, 2017a, 2018; Amand et al., 2019; Siegmund et al., 2015). Recent results show that uniform, random sampling is still challenging for configurable systems (Plazar et al., 2019). Hence, true random sampling is sometimes hard to achieve and some approximate alternatives, such as distance-based sampling, have been proposed (Kaltenecker et al., 2019).

In the context of testing software product lines, there are numerous available heuristics, algorithms, and empirical studies (Medeiros et al., 2016; Lopez-Herrejon et al., 2015; do Carmo Machado et al., 2014; Lee et al., 2012; Thüm et al., 2014). In this case, sampling is used to efficiently test a subset of configurations with the hope of finding bugs. Though the sample has not the same purpose (efficient identification of bugs), we believe a learning-based process could benefit from works done in the software testing domain. An important question is whether heuristics or sampling criterion devised originally for testing have the same efficiency when applied for learning configuration spaces.

A striking challenge is to investigate whether there exists a one-size-fits-all strategy for efficiently sampling a configuration space. The effectiveness of a sampling strategy may depend on the choice of a learning algorithm (Grebhahn et al., 2019), which is hard to anticipate. Besides, recent results suggest there is no single dominant sampling even for the same configurable system. Instead, different strategies perform best on different workloads and NFPs (Pereira et al., 2020; Grebhahn et al., 2019). An open issue is to automatically and *a priori* select the most effective sampling in a given situation. Finally, we observe that most of the reported techniques only use as input a model of the configuration space (*e.g.,* a feature model). The incorporation of knowledge and other types of input data in the sampling phase needs more investigation in future research (*e.g.,* system documentation, implementation artifacts, code smells, and system evolution).

5.5. Cost-effectiveness of learning-based solutions

A threat to practical adoption is the cost of learning-based approaches. In addition to the cost of training a machine learning model (see the earlier discussion on trade-offs), the cost of gathering data and measuring configurations can be significant (several days). Most of the NFPs reported in the literature (see Fig. 6 and Table C.8) are quantitative and related to performance. An organization may not have the resources to instrument a large and accurate campaign of performance measurements.

A general problem is to reduce the cost of measuring many configurations. For example, the use of a distributed infrastructure (*e.g.,* cloud) has the merit of offering numerous computing resources at the expense of networking issues and heterogeneity (Leitner and Cito, 2016; Bao et al., 2018). Though some countermeasures can be considered, it remains an important practical problem. Hence, there is a tension between the accuracy of measurements and the need to scale up the process to thousands of configurations. At the opposite end of the spectrum, simulation and static analysis are less costly but may approximate the real measures: few studies explore them which demand further investigation. Another issue is how performance measurements transfer to a computational environment (*e.g.,* hardware). The recent rise of transfer learning techniques is an interesting direction, but much more research is needed.

Besides, only a few works deal with uncertainty when learning configuration spaces (*e.g.,* Jamshidi et al. (2017b)). There are questions we could ask, such as how many times do we need to repeat measurements? To what degree can we trust the results? The measurement of performance is subject to intensive research: there can be bias, noise or uncertainty (Aleti et al., 2018; Colmant et al., 2018; Trubiani and Apel, 2019) that can have in turn an impact on the effectiveness of the whole learning process.

Ideally, the investments required to sample, measure, and learn configuration spaces should be outweighed by the benefits of effectively performing a task (*e.g.,* finding an optimal configuration). The reduction of the cost is a natural research direction. Another is to find a reasonable tradeoff.

5.6. Realistic evaluation

When transferring techniques from early research into practice, we have to be careful with respect to things that can go wrong and effort that is required for the different activities, *e.g.,* data acquisition and preparation, feature engineering, selecting/adapting algorithms and models, evaluation in a controlled experimental setting, deployment for use in practice, and evolution/maintenance. Looking at many papers in ML and related fields, one might get the impression that they focus on activities "in the middle", *e.g.,* the development and evaluation of algorithms, since they can be performed *in vitro* (in a controlled lab environment), whereas activities at the beginning or the end need to be done *in vivo* (in "dirty" real-life). Also, see the discussion by Bosch et al. (2020).

This preliminary hypothesis needs further investigation, but it seems that research should move out of the comfort zone in the lab and face the "dirty" reality. Some examples of works that go into this direction are given in the following.

A first step from theoretical research towards applicability is to go beyond reporting bare evaluation metrics (*e.g.,* accuracy) and at least *discuss the applicability of the presented approach*. As an example, consider Weckesser et al. (2018) who discuss the applicability and effectiveness of their approach for dynamic reconfiguration with a real-world scenario.

This can be taken further by performing the evaluation (or parts of it) in on a real system. For instance, instead of evaluating a control algorithm with theoretical considerations one can deploy that algorithm onto real physical systems which then have to face the challenges and imperfections of reality. As an example, see Jamshidi et al. (2019) who deal with the self-adaptation of autonomous robots. They first evaluate their approach in a theoretical setting, which is validated on real systems (*theoretical evaluation grounded in practice*), and second in a robotics scenario deployed onto robots (*evaluation in practice*).

Another example is work by Temple et al. (2016) where the use of machine learning was driven by an open practical problem (how to specify constraints of a product line, how to explore a large configuration space with automated support) and obtained results (rules/constraints) where validated with developers, i.e., through *collaboration with domain experts*.

A related ingredient for applicable research is to *identify and describe problems with a practical perspective* and with *input from domain experts*, ideally with input from actual application scenarios. As an example consider the explicit problem description (of modeling performance as influenced by configuration options) by Kolesnikov et al. (2018) which is based on a discussion with domain experts (for high-performance computing). The authors complete the paper with an appendix where they discuss the most influential configuration options and link that back to the domain knowledge.

As outlined in the beginning of this section, there is still a gap between research and practice. Some papers in our literature

Table A.6
Sample methods reported in the literature.

Sample method	Reference
Random	Guo et al. (2013, 2017), Jamshidi et al. (2017b,a), Oh et al. (2017), Nair et al. (2018), Temple et al. (2016, 2017a), Valov et al. (2015), Weckesser et al. (2018), Zhang et al. (2015), Acher et al. (2018), Jamshidi et al. (2019), Temple et al. (2018), Siegmund et al. (2015), Valov et al. (2017), Nair et al. (2017, 2018), Kaltenecker et al. (2019), Siegmund et al. (2017), Alipourfard et al. (2017), Siegmund et al. (2008), Zhang et al. (2016), Thornton et al. (2013), Švogor et al. (2019), Grebhahn et al. (2017), Chen et al. (2009), Osogami and Kato (2007)
Knowledge-wise heuristic	Siegmund et al. (2011, 2013b, 2017, 2008), Xu et al. (2008)
Feature-coverage heuristic	Kaltenecker et al. (2019), Sarkar et al. (2015), Siegmund et al. (2012a, 2011, 2013b, 2015), Yilmaz et al. (2014, 2006), Lillack et al. (2013), Grebhahn et al. (2017)
Feature-frequency heuristic	Sarkar et al. (2015), Siegmund et al. (2012a, 2015, 2011, 2013b, 2012b), Lillack et al. (2013), Grebhahn et al. (2017), Etxeberria et al. (2014)
Family-based simulation	Siegmund et al. (2013a)
Multi-start local search	Hutter et al. (2011)
Plackett–Burman design	Siegmund et al. (2015), Grebhahn et al. (2017)
central composite design	Grebhahn et al. (2017)
D-optimal design	Grebhahn et al. (2017)
Breakdown	Westermann et al. (2012)
Sequence type trees	Krismayer et al. (2017)
East–west sampling	Nair et al. (2018)
Exemplar sampling	Nair et al. (2018)
Constrained-driven sampling	Gargantini et al. (2017)
Diameter uncertainty strategy	Zuluaga et al. (2016)
Historical configurations	Van Aken et al. (2017a), Saleem et al. (2015)
Latin hypercube sampling	Xi et al. (2012), Jamshidi and Casale (2016), Bao et al. (2018)
Neighborhood sampling	Porter et al. (2007), Jamshidi et al. (2018)
Input-based clustering	Ding et al. (2015)
Distance-based sampling	Kaltenecker et al. (2019), Ghamizi et al. (2019)
Genetic sampling	Martinez et al. (2018), Jamshidi and Casale (2016), Safdar et al. (2017)
Interaction tree discovery	Song et al. (2013)
Arbitrarily chosen	Chen et al. (2005), Murwantara et al. (2014), Sincero et al. (2010), Samreen et al. (2016), Amand et al. (2019), Queiroz et al. (2016), Duarte et al. (2018), Chen et al. (2009)

review show it is possible to apply different research methods in order to assess the actual potential of learning-based solutions for engineering real-world systems.

5.7. Integrated tools

We observe that several studies provide either open source or publicly available implementations, mainly on top of data science libraries. Tools could assist end-users during, for instance, the configuration process and the choice of options. Variability management tools could be extended to encompass the learning stages reported in this SLR. It could also benefit to software developers in charge of maintaining configurable systems. However, none of the reported studies provide tool support that is fully integrated into existing SPL platforms or mainstream software tools. An exception is SPLConqueror (Siegmund et al., 2012b) that supports an interactive configuration process. We believe the integration of tooling support is key for practical adoption and the current lack may partly explain the gap between practice and research.

5.8. Final remarks and summary

The problem of learning configuration spaces is at the intersection of artificial intelligence (mainly statistical ML and constraint reasoning) and software engineering. There are many existing works that do not specifically address software systems and are yet related. Hence, there is considerable potential to "connect the dots" and reuse state-of-the-art methods for learning highly-dimensional spaces like software configuration spaces.

For instance, an active research area that is related to choosing the best algorithm is the automated algorithm selection problem (Kerschke et al., 2019) where given a set of algorithms, and a specific problem instance to be solved, the problem is to determine which of the algorithms can be selected to perform best on that instance. In our case, the set comprises all (valid) configurations of a single, parameterized algorithm (whereas the set of algorithms come from different software implementation and systems for the problem of algorithm selection). We also believe the line of research work presented in this literature review can be related to specific problems like compiler autotuning (see *e.g.*, Ashouri et al. (2018)) or database management system tuning (see *e.g.*, Van Aken et al. (2017b)).

We encourage researchers to explore possible synergies, *e.g.*, how solutions and results in both areas can be reused or adapted.

Finally, Table 5 summarizes the challenges discussed in this section with (1) points to consider in practical applications and (2) opportunities for further research.

6. Threats to validity

This section discusses potential threats to validity that might have affected the results of the SLR. We faced similar threats to validity as any other SLR. The findings of this SLR may have been affected by bias in the selection of the primary studies, inaccuracy in the data extraction and in the classification of the primary studies, and incompleteness in defining the open challenges. Next, we summarize the main threats to the validity of our work and the strategies we have followed to minimize their impact. We discussed the SLR validity with respect to the two groups of common threats to validity: *internal* and *external* validity (Wohlin et al., 2000).

Internal validity. An internal validity threat concerns the reliability of the selection and data extraction process. To further

Table B.7
Learning techniques reported in the literature. *A1*: Pure prediction; *A2*: Interpretability of configurable systems; *A3*: Optimization; *A4*: Dynamic configuration; *A5*: Mining constraints; *A6*: SPL evolution.

Reference	Learning technique	Applicability
Yilmaz et al. (2014)	Adaptive ELAs, Multi-Class FDA-CIT, Static Error Locating Arrays (ELAs), Ternary-Class FDA-CIT, Test Case-Aware CIT, Traditional CIT	*A1*
Valov et al. (2015)	Bagging	*A1*
Zhang et al. (2015)	Fourier Learning of Boolean Functions	*A1*
Kolesnikov et al. (2017)	Frequent Item Set Mining	*A1*
Siegmund et al. (2013a)	Graph Family-Based Variant Simulator	*A1*
Couto et al. (2017)	Implicit Path Enumeration Technique (IPET)	*A1*
Queiroz et al. (2016)	Naive Bayes	*A1*
Jamshidi et al. (2017a), Kolesnikov et al. (2018), Siegmund et al. (2015), Weckesser et al. (2018), Kolesnikov et al. (2017), Kaltenecker et al. (2019), Jamshidi et al. (2019), Siegmund et al. (2008)	Step-Wise Multiple Linear Regression	*A1*, *A2*, *A3*, *A4*
Guo et al. (2013, 2017), Jamshidi et al. (2017a), Murwantara et al. (2014), Nair et al. (2017, 2018), Sarkar et al. (2015), Temple et al. (2016, 2017a), Valov et al. (2015, 2017), Westermann et al. (2012), Acher et al. (2018), Nair et al. (2018), Yilmaz et al. (2006), Krismayer et al. (2017), Porter et al. (2007), Zheng et al. (2007), Zhang et al. (2016), Song et al. (2013)	Classification and Regression Trees (CART)	*A1*, *A2*, *A3*, *A4*, *A5*
Siegmund et al. (2017)	Kernel Density Estimation and NSGA-II	*A1*, *A2*, *A3*, *A4*, *A5*, *A6*
Siegmund et al. (2011, 2012a,b, 2013b)	Feature's Influence Delta	*A1*, *A3*
Alipourfard et al. (2017), Westermann et al. (2012), Jamshidi et al. (2017b), Van Aken et al. (2017a), Jamshidi et al. (2018), Jamshidi and Casale (2016), Zuluaga et al. (2016)	Gaussian Process Model	*A1*, *A3*, *A4*
Murwantara et al. (2014), Samreen et al. (2016), Chen et al. (2005), Lillack et al. (2013)	Linear Regression	*A1*, *A3*, *A4*
Valov et al. (2015), Amand et al. (2019), Queiroz et al. (2016), Bao et al. (2018), Thornton et al. (2013)	Random Forest	*A1*, *A3*, *A5*
Amand et al. (2019), Temple et al. (2018), Valov et al. (2015), El Afia and Sarhani (2017)	Support Vector Machine	*A1*, *A3*, *A5*
Amand et al. (2019), Queiroz et al. (2016), Song et al. (2013)	C4.5 (J48)	*A1*, *A5*
Sincero et al. (2010), Etxeberria et al. (2014)	Covariance Analysis	*A2*
Jamshidi et al. (2017a)	Multinomial Logistic Regression	*A2*
Duarte et al. (2018)	K-Plane Algorithm	*A2*, *A4*
Saleem et al. (2015)	AdaRank	*A3*
Murwantara et al. (2014)	Bagging Ensembles of CART, Bagging Ensembles of MLPs	*A3*
Martinez et al. (2018)	Data Mining Interpolation Technique	*A3*
Van Aken et al. (2017a)	Factor Analysis, k-means, Ordinary Least Squares	*A3*
Westermann et al. (2012), Švogor et al. (2019)	Genetic Programming (GP)	*A3*
Westermann et al. (2012)	Kriging	*A3*
Ding et al. (2015)	Max-Apriori Classifier, Exhaustive Feature Subsets Classifiers, All Features Classifier, Incremental Feature Examination classifier	*A3*
Osogami and Kato (2007)	Quick Optimization via Guessing	*A3*
Hutter et al. (2011)	Random Online Adaptive Racing (ROAR), Sequential Model-based Algorithm Configuration (SMAC)	*A3*
Švogor et al. (2019)	Simulated Annealing	*A3*
Xi et al. (2012)	Smart Hill-Climbing	*A3*
Oh et al. (2017)	Statistical Recursive Searching	*A3*
Ghamizi et al. (2019)	Tensorflow and Keras	*A3*
Thornton et al. (2013)	Tree-structured Parzen Estimator (TPE)	*A3*
Samreen et al. (2016), Westermann et al. (2012), Grebhahn et al. (2017)	Multivariate Adaptive Regression Splines (MARS), Multivariate Polynomial Regression	*A3*, *A4*
Samreen et al. (2016), Xu et al. (2008)	Ridge Regression	*A3*, *A4*
Murwantara et al. (2014), Amand et al. (2019)	Multilayer Perceptrons (MLPs)	*A3*, *A5*
Chen et al. (2009)	Actor–Critic Learning	*A4*
Samreen et al. (2016)	Lasso	*A4*
Sharifloo et al. (2016)	Reinforcement Learning	*A4*, *A6*
Gargantini et al. (2017)	CitLab Model	*A5*
Temple et al. (2018)	Evasion Attack	*A5*
Amand et al. (2019)	Hoeffding Tree, K*, kNN, Logistic Model Tree, Logistic Regression, Naive Bayes, PART Decision List, Random Committee, REP Tree, RIPPER	*A5*
Safdar et al. (2017)	Pruning Rule-Based Classification (PART)	*A5*

increase the internal validity of the review results, the search for relevant studies was conducted in several relevant scientific databases, and it was focused not only on journals but also on conferences, symposiums, and workshops. Moreover, we conducted the inclusion and exclusion processes in parallel by involving three researchers and we cross-checked the outcome after each phase. In the case of disagreements, we discussed until a final decision was achieved. Furthermore, we documented potentially relevant studies that were excluded. However, the quality of the search engines could have influenced the completeness of the identified primary studies (*i.e.*, our search may have missed those studies whose authors did not use the terms we used in our search string to specify keywords).

For the selected papers, a potential threat to validity is the reliability and accuracy of the data extraction process, since not all information was obvious to extract (*e.g.*, many papers lacked details about the measurement procedure and the validation design of the reported study). Consequently, some data had to be interpreted which involved subjective decisions by the researchers. Therefore, to ensure the validity, multiple sources of data were analyzed, *i.e.*, papers, websites, technical reports, manuals, and executable. Moreover, whenever there was a doubt about some extracted data in a particular paper, we discussed the reported data from different perspectives in order to resolve all discrepancies. However, we are aware that the data extraction process is a subjective activity and likely to yield different results when executed by different researchers.

External validity. A major threat to external validity is related to the identification of primary studies. Key terms are directly related to the *scope* of the paper and they can suffer a high variation. We limited the search for studies mainly targeting software systems (*e.g.*, software product lines) and thus mainly focus on software engineering conferences. This may affect the completeness of our search results since we are aware of some studies outside the software engineering community that also address the learning of software configurable systems. To minimize this limitation and avoid missing relevant papers, we also analyzed the references of the primary studies to identify other relevant studies. In addition, this SLR was based on a strict protocol described in Section 2 which was discussed before the start of the review to increase the reliability of the selection and data extraction processes of the primary studies and allow other researchers to replicate this review.

Another external validity concerns the description of open challenges. We are aware that the completeness of open challenges is another limitation that should be considered while interpreting the results of this review. It is also important to explore general contributions from other fields outside the software domain to fully gather the spread knowledge, which may extend the list of findings in this field. Therefore, in future work, the list of findings highlighted in Section 5 may be extended by conducting an additional review, making use of other keywords to be able to find additional relevant studies outside the software community.

7. Related work

After the introduction of SLR in software engineering in 2004, the number of published reviews in this field has grown significantly (Kitchenham et al., 2009). A broad SLR has been conducted by Heradio et al. (2016) to identify the most influential researched topics in SPL, and how the interest in those topics has evolved over the years. Although these reviews are not directly related to ours, the high level of detail of their research methodology supported to structure and define our own methodology.

Benavides et al. (2010) presented the results of a literature review to identify a set of operations and techniques that provide support to the automatic analysis of variability models. In a similar scenario, Lisboa et al. (2010) and Pereira et al. (2015) conducted an SLR on variability management of SPLs. They reported several dozens of approaches and tools to support stakeholders in an interactive and automatic configuration process. Strict to the application engineering phase, Ochoa et al. (2018) and Harman et al. (2014) conducted a literature review on the use of search-based software engineering techniques for optimization of SPL configurations. In contrast to these previous reviews, our SLR provides further details on the use of automatic learning techniques to explore large configuration spaces. We contribute with a catalogue of sampling approaches, measurement procedures, learning techniques, and validation steps that serves as a summarization of the results in this specific field.

In Sayagh et al. (2018), 14 software engineering experts and 229 Java software engineers were interviewed to identify major activities and challenges related to configuration engineering. In complement, a SLR was conducted. In order to select papers, authors focused on those in which the title and abstract contain the keyword "config*", "misconfig*", or "mis-config*". On the one hand, the scope is much broader: It spans all engineering activities of configurable systems whereas our literature review specifically targets learning-based approaches and applications. On the other hand, we learn that configuration spaces are studied in many different engineering contexts and we take care of considering a broader terminology (see Table 1). Despite different scope and reviewing methodology, the two surveys are complementary. A research direction is to further explore how learning-based approaches classified in this literature review could support configuration engineering activities identified in Sayagh et al. (2018).

In the context of software fault prediction, Malhotra (2015) conducts an SLR to summarize the learning techniques used in this field and the performance accuracy of these techniques. They classify machine learning techniques into seven categories: Decision Tree, Bayesian Learning, Ensemble Learning, Rule Based Learning, Evolutionary Algorithms, Neural Networks and Miscellaneous. Although our SLR is much restricted by considering only primary studies on the field of software product lines, we also consider fault as a performance metric and thus we also report on most of these techniques. Also, our results depict that the evaluation metrics "precision, recall, and F-Measure" are the most commonly used performance measures to compute the accuracy of reported classification techniques. In addition, our SLR is much broad once it also considers other properties besides fault, therefore we report on both classification and regression techniques and evaluation metrics.

On the context of testing and verifying a software product line, several works (Varshosaz et al., 2018; Medeiros et al., 2016; Lopez-Herrejon et al., 2015; do Carmo Machado et al., 2014; Lee et al., 2012; Thüm et al., 2014) discussed product sampling techniques. They classify the proposed techniques into different categories and discuss the required input and criteria used to evaluate these techniques, such as the sample size, the rate of fault detection, and the tool support. Future work could benefit from their results through the use of sampling techniques still not explored by learning techniques (*e.g.* code and requirements coverage), as well as the SPL testing community could benefit from the sampling techniques reported in this paper still not used for testing. In Khalil Abbasi et al. (2013), 111 real-world Web configurators are analyzed but do not consider learning mechanisms.

None of surveys mentioned above directly address the use of learning techniques to explore the behavior of large configuration spaces through performance prediction. The main topics of previous SLRs include variability model analysis, variability management, configuration engineering, fault prediction and SPL testing. There are several reviews investigating the use of learning-based techniques in many other scenarios, such as spam filtering (Guzella and Caminhas, 2009; Crawford et al., 2015), text-documents classification (Khan et al., 2010), automated planning (Jiménez et al., 2012), genomic medicine (Leung

Table C.8
Subject systems supported in the literature.

Reference	Name	Domain	Non-functional properties
Amand et al. (2019)	Thingiverse's	3D printer	defects
Xi et al. (2012)	IBM WebSphere	Application server	throughput
Guo et al. (2017), Kolesnikov et al. (2018), Siegmund et al. (2015)	Clasp	ASP solver	response time
Zuluaga et al. (2016)	SNW	Asset management	area and throughput
Ding et al. (2015)	Binpacking	Binpacking algorithm	execution time and accuracy
Jamshidi et al. (2018)	XGBoost	Boosting algorithms	training time
Sharifloo et al. (2016)	SaaS system	Cloud computing	response time
Ding et al. (2015)	Clustering	Clustering algorithm	execution time and accuracy
Guo et al. (2017), Kolesnikov et al. (2018), Siegmund et al. (2015, 2013a)	AJStats	Code analyzer	response time
Nair et al. (2018), Jamshidi et al. (2017a), Siegmund et al. (2015), Jamshidi et al. (2018)	SaC	Code analyzer	I/O time, response time
Kaltenecker et al. (2019)	POLLY	Code optimizer	runtime
Gargantini et al. (2017)	Libssh	Combinatorial model	defects
Gargantini et al. (2017)	Telecom	Communication system	defects
Guo et al. (2013, 2017), Oh et al. (2017), Kolesnikov et al. (2018), Nair et al. (2018), Sarkar et al. (2015), Siegmund et al. (2012a, 2013b), Temple et al. (2017a), Siegmund et al. (2015), Zhang et al. (2015), Nair et al. (2018), Kaltenecker et al. (2019), Zuluaga et al. (2016), Zhang et al. (2016)	LLVM	Compiler	memory footprint, performance, response time, code complexity, compilation time
Lillack et al. (2013)	Compressor SPL	Compression library	compression time, memory usage and compression ratio
Kaltenecker et al. (2019)	7Z	Compression library	Compression time
Siegmund et al. (2015), Guo et al. (2017), Kolesnikov et al. (2018), Nair et al. (2017), Kaltenecker et al. (2019)	LRZIP	Compression library	compressed size, compression time, compilation time
Siegmund et al. (2013b)	RAR	Compression library	code complexity
Valov et al. (2017)	XZ	Compression library	compression time
Siegmund et al. (2011, 2012b, 2013b,a)	ZipMe	Compression library	memory footprint, performance, code complexity, time
Murwantara et al. (2014)	WordPress	Content management	CPU power consumption
Siegmund et al. (2011, 2012b, 2013b, 2008)	LinkedList	Data structures	memory footprint, performance, maintainability, binary size
Siegmund et al. (2013b)	Curl	Data transfer	code complexity
Siegmund et al. (2013b), Nair et al. (2017)	Wget	Data transfer	memory footprint, code complexity
Van Aken et al. (2017a)	Actian Vector	Database system	runtime
Jamshidi et al. (2017b)	Apache Cassandra	Database system	latency
Guo et al. (2013, 2017), Oh et al. (2017), Kolesnikov et al. (2018), Nair et al. (2017), Sarkar et al. (2015), Siegmund et al. (2012a, 2013b), Temple et al. (2017a), Siegmund et al. (2011, 2012b, 2015), Zhang et al. (2015), Nair et al. (2018), Kaltenecker et al. (2019), Siegmund et al. (2008), Zhang et al. (2016)	Berkeley DB	Database system	I/O time, memory footprint, performance, response time, code complexity, maintainability, binary size
Siegmund et al. (2008)	FAME-DBMS	Database system	maintainability, binary size, performance
Yilmaz et al. (2014), Van Aken et al. (2017a), Zheng et al. (2007), Song et al. (2013)	MySQL	Database system	defects, throughput, latency
Van Aken et al. (2017a)	Postgres	Database system	throughput, latency
Siegmund et al. (2011, 2012b, 2013b)	Prevayler	Database system	memory footprint, performance
Guo et al. (2013, 2017), Nair et al. (2017), Sarkar et al. (2015), Siegmund et al. (2012a, 2013b), Temple et al. (2017a), Siegmund et al. (2011, 2012b, 2013b), Jamshidi et al. (2017a), Valov et al. (2017), Nair et al. (2018), Kolesnikov et al. (2017)	SQLite	Database system	memory footprint, performance, response time, code complexity, runtime
Chen et al. (2005)	StockOnline	Database system	response time
Bao et al. (2018)	Kafka	Distributed systems	throughput
Ghamizi et al. (2019)	DNN	DNNs algorithms	accuracy of predictions
Acher et al. (2018)	Curriculum vitae	Document	number of pages
Acher et al. (2018)	Paper	Document	number of pages
Duarte et al. (2018)	RUBiS	E-commerce	response time
Siegmund et al. (2013a)	EMAIL	E-mail client	time

(continued on next page)

Table C.8 (*continued*).

Reference	Name	Domain	Non-functional properties
Kolesnikov et al. (2017)	MBED TLS	Encryption library	response time
Westermann et al. (2012)	SAP ERP	Enterprise Application	response time
Nair et al. (2018)	noc-CM-log	FPGA	CPU power consumption, runtime
Nair et al. (2018)	sort-256	FPGA	area, throughput
Etxeberria et al. (2014)	E-Health System	Health	response time
Guo et al. (2017), Siegmund et al. (2015), Temple et al. (2017a), Kaltenecker et al. (2019)	HIPA^{cc}	Image processing	response time
Couto et al. (2017)	Disparity SPL	Image processing	energy consumption
Siegmund et al. (2011, 2012b, 2013b)	PKJab	Instant messenger	memory footprint, performance
Hutter et al. (2011)	IBM ILOG	Integer solver	runtime
Westermann et al. (2012)	SPECjjbb2005	Java Server	response time, throughput
Thornton et al. (2013)	WEKA	Learning algorithm	accuracy of predictions
Ding et al. (2015)	SVD	Linear algebra	execution time and accuracy
Nair et al. (2018)	Trimesh	Mesh solver	iterations, response time
Siegmund et al. (2013a)	MBENCH	Micro benchmark	time
Yilmaz et al. (2006), Porter et al. (2007)	ACE+TAO system	Middleware software	defects
Siegmund et al. (2011, 2012b, 2013b)	SensorNetwork	Network simulator	memory footprint, performance
Weckesser et al. (2018)	Simonstrator	Network simulator	latency
Zuluaga et al. (2016)	NoC	Network-based system	energy and runtime
Ding et al. (2015)	Helmholtz 3D	Numerical analysis	execution time and accuracy
Ding et al. (2015)	Poisson 2D	Numerical analysis	execution time and accuracy
Sincero et al. (2010), Siegmund et al. (2011, 2012b, 2013b)	Linux kernel	Operating system	memory footprint, performance
Jamshidi et al. (2018)	DNN	Optimization algorithm	response time
Martinez et al. (2018)	Art system	Paint	user likeability
Grebhahn et al. (2017)	Multigrid system	Equations solving	time of each interaction
Jamshidi et al. (2017b)	CoBot System	Robotic system	CPU usage
Siegmund et al. (2015), Temple et al. (2017a), Kaltenecker et al. (2019)	JavaGC	Runtime environment	response time
Švogor et al. (2019)	Robot	Runtime environment	energy consumption and execution time
El Afia and Sarhani (2017), Xu et al. (2008)	Hand	SAT solver	runtime
El Afia and Sarhani (2017), Xu et al. (2008)	Indu	SAT solver	runtime
El Afia and Sarhani (2017), Xu et al. (2008)	Rand	SAT solver	runtime
Hutter et al. (2011)	SAPS	SAT solver	runtime
Jamshidi et al. (2017a), Hutter et al. (2011)	SPEAR	SAT solver	runtime, response time
Saleem et al. (2015)	QWS dataset	Services	availability, throughput, successability, reliability, compliance, best practice, documentation
Queiroz et al. (2016)	BusyBox	Software suite	defect, process metrics
Krismayer et al. (2017)	Plant automation	Software-intensive SoS	defects
Ding et al. (2015)	Sort	Sort algorithm	execution time
Kolesnikov et al. (2018), Nair et al. (2017), Temple et al. (2017a), Siegmund et al. (2015), Kaltenecker et al. (2019)	DUNE	Stencil code	response time
Kolesnikov et al. (2018), Nair et al. (2017), Siegmund et al. (2015), Temple et al. (2017a)	HSMGP	Stencil code	response time, runtime
Nair et al. (2017), Jamshidi et al. (2017b), Nair et al. (2018), Zhang et al. (2015), Jamshidi and Casale (2016), Jamshidi et al. (2018), Zhang et al. (2016)	Apache	Stream processing	latency, throughput, performance
Gargantini et al. (2017)	Concurrency	Testing problem	defects
Samreen et al. (2016)	VARD on EC2	Text manager	response time
Gargantini et al. (2017)	Aircraft	Toy example	defects
Siegmund et al. (2013a)	ELEVATOR	Toy example	time
Gargantini et al. (2017)	WashingMachine	Toy example	defects
Siegmund et al. (2011, 2012b, 2013b)	Violet	UML editor	memory footprint, performance

(*continued on next page*)

et al., 2015), electricity load forecasting (Yildiz et al., 2017), and others. Overall, these works provide us with a large dataset of sampling, learning, and validation techniques that can be further explored/adapted in the field of SPLs.

8. Conclusion

We presented a systematic literature review related to the use of learning techniques to analyze large configuration software spaces. We analyzed the literature in terms of a four-stage

Table C.8 (*continued*).

Reference	Name	Domain	Non-functional properties
Temple et al. (2016, 2017a, 2018)	MOTIV	Video encoder	video quality
Kaltenecker et al. (2019)	VP9	Video encoder	encoding time
Guo et al. (2013, 2017), Oh et al. (2017), Kolesnikov et al. (2018), Nair et al. (2017), Sarkar et al. (2015), Siegmund et al. (2012a, 2013b), Nair et al. (2018), Jamshidi et al. (2017a), Temple et al. (2017a), Siegmund et al. (2015), Zhang et al. (2015), Valov et al. (2017), Nair et al. (2018), Kaltenecker et al. (2019), Zhang et al. (2016)	x264	Video encoder	CPU power consumption, encoding time, Peak Signal to Noise Ratio, response time, code complexity, video quality, performance
Temple et al. (2017a)	OpenCV	Video tracking	performance
Safdar et al. (2017)	C60 and MX300	Virtual environment	defects
Alipourfard et al. (2017)	Amazon EC2	Web cloud service	performance
Guo et al. (2013, 2017), Oh et al. (2017), Kolesnikov et al. (2018), Nair et al. (2017), Sarkar et al. (2015), Siegmund et al. (2012a), Temple et al. (2017a), Siegmund et al. (2015), Nair et al. (2018), Yilmaz et al. (2014), Zheng et al. (2007)	Apache	Web server	response rate, response time, workload, defects, throughput
Osogami and Kato (2007)	Stock Brokerage	Web system	throughput, response time
Song et al. (2013)	vsftpd	FTP daemon	defects
Song et al. (2013)	ngIRCd	IRC daemon	defects

process: "sampling, measuring, learning, and validation" (see Section 3). Our contributions are fourfold. First, we identified the application of each approach which can guide researchers and industrial practitioners when searching for an appropriate technique that fits their current needs. Second, we classified the literature with respect to each learning stage. Mainly, we give an in-depth view of *(i)* sampling techniques and employed design; *(ii)* employed learning techniques; *(iii)* measurement properties and effort for measurement; and *(iv)* how learning techniques are empirically validated. Third, we identify the main shortcomings of existing approaches and non-addressed research areas to be explored by future work. Fourth, we provide a catalog of subject systems used in the literature together with their application domains and qualitative or quantitative properties of interest. We welcome any contribution by the community: all the material can be found in our supplementary Website (Pereira et al., 2019).

Our results reveal that the research in this field is application-dependent: Though the overall scheme remains the same ("sampling, measuring, learning"), the concrete choice of techniques should trade various criterion like safety, cost, accuracy, and interpretability. The proposed techniques have typically been validated with respect to different metrics depending on their tasks (*e.g.*, performance prediction). Although the results are quite accurate, there is still need to decrease learning errors or to generalize predictions to multiple computing environments. Given the increasing interest and importance of this field, there are many exciting opportunities of research at the interplay of artificial intelligence and software engineering.

CRediT authorship contribution statement

Juliana Alves Pereira: Conceptualization, Methodology, Writing – original draft, Data curation, Validation, Writing – review & editing. **Mathieu Acher:** Supervision, Conceptualization, Methodology, Writing – original draft, Data curation, Validation, Writing – review & editing. **Hugo Martin:** Conceptualization, Methodology, Writing – review & editing. **Jean-Marc Jézéquel:** Supervision, Writing – review & editing. **Goetz Botterweck:** Methodology, Validation, Writing – review & editing. **Anthony Ventresque:** Methodology, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was partially funded by the ANR-17-CE25-0010-01 VaryVary project and by Science Foundation Ireland grant 13/RC/2094. We would like to thank Paul Temple for his early comments on a draft of this article.

Appendix A. Sampling methods

Table A.6 shows the sample methods adopted by each study. The first column is about the method and the second column identifies the study reference(s). There are 23 high-level sample methods documented in the literature.

Appendix B. Learning techniques

Table B.7 sketches which learning techniques are supported in the literature and what application objective they address. The first column identifies the study reference(s). The second and third columns identify the name of the learning technique and its application objective (see Section 4.1), respectively. (Notice that the application objective is related to the scenarios in which each learning technique has been already used in the literature; it means some learning techniques could well be applied for other scenarios in the future).

Appendix C. Configurable systems

Table C.8 presents all the subject systems used in the literature together with NFPs. The first column identifies the reference(s). The second and third columns describe the name and domain of the system, respectively. The fourth column points out the measured NFP(s).

References

Acher, M., Collet, P., Lahire, P., France, R.B., 2013. Familiar: A domain-specific language for large scale management of feature models. Sci. Comput. Program. 78 (6), 657–681.

Acher, M., Martin, H., Pereira, J.A., Blouin, A., Eddine Khelladi, D., Jézéquel, J.-M., 2019a. Learning from Thousands of Build Failures of Linux Kernel Configurations. Technical report, Inria; IRISA, URL https://hal.inria.fr/hal-02147012.

Acher, M., Martin, H., Pereira, J.A., Blouin, A., Jézéquel, J.-M., Khelladi, D.E., Lesoil, L., Barais, O., 2019b. Learning Very Large Configuration Spaces: What Matters for Linux Kernel Sizes. Research report, Inria Rennes - Bretagne Atlantique, URL https://hal.inria.fr/hal-02314830.

Acher, M., Temple, P., Jezequel, J.-M., Galindo, J.A., Martinez, J., Ziadi, T., 2018. Varylatex: Learning paper variants that meet constraints. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems. ACM, pp. 83–88.

Akers, S.B., 1978. Binary decision diagrams. IEEE Trans. Comput. (6), 509–516.

Aleti, A., Trubiani, C., van Hoorn, A., Jamshidi, P., 2018. An efficient method for uncertainty propagation in robust software performance estimation. J. Syst. Softw. 138, 222–235. http://dx.doi.org/10.1016/j.jss.2018.01.010.

Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M., 2017. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), pp. 469–482.

Amand, B., Cordy, M., Heymans, P., Acher, M., Temple, P., Jézéquel, J.-M., 2019. Towards learning-aided configuration in 3d printing: Feasibility study and application to defect prediction. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems. ACM, p. 7.

Apel, S., Batory, D., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer-Verlag.

Ashouri, A.H., Killian, W., Cavazos, J., Palermo, G., Silvano, C.,

Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wasowski, A., 2016. Clafer: unifying class and feature modeling. Softw. Syst. Model. 15 (3), 811–845.

Bao, L., Liu, X., Xu, Z., Fang, B., 2018. Autoconfig: automatic configuration tuning for distributed message systems. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). ACM, pp. 29–40.

Benavides, D., Martín-Arroyo, P.T., Cortés, A.R., 2005. Automated reasoning on feature models. In: International Conference on Advanced Information Systems Engineering (CAiSE), Vol. 5. Springer, pp. 491–503.

Benavides, D., Segura, S., Ruiz-Cortés, A., 2010. Automated analysis of feature models 20 years later: a literature review. Inf. Syst. 35 (6), 615–708.

Bosch, J., Crnkovic, I., Olsson, H.H., 2020. Engineering ai systems: A research agenda. arXiv:2001.07522.

Cashman, M., Cohen, M.B., Ranjan, P., Cottingham, R.W., 2018. Navigating the maze: the impact of configurability in bioinformatics software. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 757–767. http://dx.doi.org/10.1145/3238147.3240466, URL http://doi.acm.org/10.1145/3238147.3240466.

Chen, H., Jiang, G., Zhang, H., Yoshihira, K., 2009. Boosting the performance of computing systems through adaptive configuration tuning. In: ACM Symposium on Applied Computing (SAC). ACM, pp. 1045–1049.

Chen, S., Liu, Y., Gorton, I., Liu, A., 2005. Performance prediction of component-based applications. J. Syst. Softw. 74 (1), 35–43.

Colmant, M., Rouvoy, R., Kurpicz, M., Sobe, A., Felber, P., Seinturier, L., 2018. The next 700 CPU power models. J. Syst. Softw. 144, 382–396. http://dx.doi.org/10.1016/j.jss.2018.07.001.

Couto, M., Borba, P., Cunha, J., Fernandes, J.P., Pereira, R., Saraiva, J., 2017. Products go green: Worst-case energy consumption in software product lines. In: Proceedings of the 21st International Systems and Software Product Line Conference-Volume A. ACM, pp. 84–93.

Crawford, M., Khoshgoftaar, T.M., Prusa, J.D., Richter, A.N., Al Najada, H., 2015. Survey of review spam detection using machine learning techniques. J. Big Data 2 (1), 23.

Ding, Y., Ansel, J., Veeramachaneni, K., Shen, X., O'Reilly, U.-M., Amarasinghe, S., 2015. Autotuning algorithmic choice for input sensitivity. In: ACM SIGPLAN Notices, Vol. 50. ACM, pp. 379–390.

do Carmo Machado, I., Mcgregor, J.D., Cavalcanti, Y.C., De Almeida, E.S., 2014. On strategies for testing software product lines: A systematic literature review. Inf. Softw. Technol. 56 (10), 1183–1199.

Duarte, F., Gil, R., Romano, P., Lopes, A., Rodrigues, L., 2018. Learning non-deterministic impact models for adaptation. In: Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems. ACM, pp. 196–205.

Eichelberger, H., Qin, C., Sizonenko, R., Schmid, K., 2016. Using ivml to model the topology of big data processing pipelines. In: International Systems and Software Product Line Conference (SPLC). ACM, pp. 204–208.

El Afia, A., Sarhani, M., 2017. Performance prediction using support vector machine for the configuration of optimization algorithms. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech). IEEE, pp. 1–7.

Etxeberria, L., Trubiani, C., Cortellessa, V., Sagardui, G., 2014. Performance-based selection of software and hardware features under parameter uncertainty. In: Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures. ACM, pp. 23–32.

Gargantini, A., Petke, J., Radavelli, M., 2017. Combinatorial interaction testing for automated constraint repair. In: 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, pp. 239–248.

Ghamizi, S., Cordy, M., Papadakis, M., Traon, Y.L.,

Grebhahn, A., Rodrigo, C., Siegmund, N., Gaspar, F.J., Apel, S., 2017. Performance-influence models of multigrid methods: A case study on triangular grids. Concurr. Comput.: Pract. Exper. 29 (17), e4057.

Grebhahn, A., Siegmund, N., Apel, S., 2019. Predicting performance of software configurations: There is no silver bullet. arXiv:1911.12643.

Guo, J., Czarnecki, K., Apel, S., Siegmund, N., Wasowski, A., 2013. Variability-aware performance prediction: A statistical learning approach. In: ASE.

Guo, J., White, J., Wang, G., Li, J., Wang, Y., 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. J. Syst. Softw. 84 (12), 2208–2221.

Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., Yu, H., 2017. Data-efficient performance learning for configurable systems. Empir. Softw. Eng. 1–42.

Guzella, T.S., Caminhas, W.M., 2009. A review of machine learning approaches to spam filtering. Expert Syst. Appl. 36 (7), 10206–10222.

Halin, A., Nuttinck, A., Acher, M., Devroey, X., Perrouin, G., Baudry, B., 2019. Test them all, is it worth it? assessing configuration sampling on the jhipster web development stack. Empir. Softw. Eng. http://dx.doi.org/10.1007/s10664-018-9635-4.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The weka data mining software: an update. ACM SIGKDD Explor. Newsl. 11 (1), 10–18.

Hallsteinsen, S.O., Hinchey, M., Park, S., Schmid, K., 2008. Dynamic software product lines. IEEE Comput. 41 (4), 93–95. http://dx.doi.org/10.1109/MC.2008.123.

Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J., Zhang, Y., 2014. Search based software engineering for software product line engineering: a survey and directions for future work. In: Proceedings of the 18th International Software Product Line Conference-Volume 1. ACM, pp. 5–18.

Heradio, R., Perez-Morago, H., Fernandez-Amoros, D., Cabrerizo, F.J., Herrera-Viedma, E., 2016. A bibliometric analysis of 20 years of research on software product lines. Inf. Softw. Technol. 72, 1–15.

Hoda, R., Salleh, N., Grundy, J., Tee, H.M., 2017. Systematic literature reviews in agile software development: A tertiary study. Inf. Softw. Technol. 85, 60–70.

Hutter, F., Hoos, H.H., Leyton-Brown, K., 2011. Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization. Springer, pp. 507–523.

Jamshidi, P., Cámara, J., Schmerl, B., Kästner, C., Garlan, D.,

Jamshidi, P., Casale, G., 2016. An uncertainty-aware approach to optimal configuration of stream processing systems. In: 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, pp. 39–48.

Jamshidi, P., Siegmund, N., Velez, M., Kästner, C., Patel, A., Agarwal, Y., 2017a. Transfer learning for performance modeling of configurable systems: an exploratory analysis. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE Press, pp. 497–508, URL http://dl.acm.org/citation.cfm?id=3155625.

Jamshidi, P., Velez, M., Kästner, C., Siegmund, N., 2018. Learning to sample: exploiting similarities across environments to learn performance models for configurable systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 71–82.

Jamshidi, P., Velez, M., Kästner, C., Siegmund, N., Kawthekar, P., 2017b. Transfer learning for improving model predictions in highly configurable software. In: 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2017, Buenos Aires, Argentina, May 22–23, 2017. pp. 31–41. http://dx.doi.org/10.1109/SEAMS.2017.11.

Jiménez, S., De La Rosa, T., Fernández, S., Fernández, F., Borrajo, D., 2012. A review of machine learning for automated planning. Knowl. Eng. Rev. 27 (4), 433–467.

Kaltenecker, C., Grebhahn, A., Siegmund, N., Guo, J., Apel, S., 2019. Distance-based sampling of software configuration spaces. In: Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE). ACM.

Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S., 1990. Feature-Oriented Domain Analysis (FODA). Tech. Rep. CMU/SEI-90-TR-21, SEI.

Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.,

Khalil Abbasi, E., Hubaux, A., Acher, M., Boucher, Q., Heymans, P., 2013. The anatomy of a sales configurator: An empirical study of 111 cases. In: CAiSE'13.

Khan, A., Baharudin, B., Lee, L.H., Khan, K., 2010. A review of machine learning algorithms for text-documents classification. J. Adv. Inf. Technol. 1 (1), 4–20.

Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering–a systematic literature review. Inf. Softw. Technol. 51 (1), 7–15.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Citeseer.

Kolesnikov, S., Siegmund, N., Kästner, C., Apel, S.,

Kolesnikov, S., Siegmund, N., Kästner, C., Grebhahn, A., Apel, S., 2018. Tradeoffs in modeling performance of highly configurable software systems. Softw. Syst. Model. 1–19.

Krismayer, T., Rabiser, R., Grünbacher, P., 2017. Mining constraints for event-based monitoring in systems of systems. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE Press, pp. 826–831.

Lee, J., Kang, S., Lee, D., 2012. A survey on software product line testing. In: Proceedings of the 16th International Software Product Line Conference-Volume 1. ACM, pp. 31–40.

Leitner, P., Cito, J., 2016. Patterns in the chaos - A study of performance variation and predictability in public iaas clouds. ACM Trans. Internet Tech. 16 (3), 15:1–15:23. http://dx.doi.org/10.1145/2885497.

Leung, M.K., Delong, A., Alipanahi, B., Frey, B.J., 2015. Machine learning in genomic medicine: a review of computational problems and data sets. Proc. IEEE 104 (1), 176–197.

Lillack, M., Müller, J., Eisenecker, U.W., 2013. Improved prediction of non-functional properties in software product lines with domain context. Softw. Eng..

Lisboa, L.B., Garcia, V.C., Lucrédio, D., de Almeida, E.S., de Lemos Meira, S.R., de Mattos Fortes, R.P., 2010. A systematic review of domain analysis tools. Inf. Softw. Technol. 52 (1), 1–13.

Lopez-Herrejon, R.E., Fischer, S., Ramler, R., Egyed, A., 2015. A first systematic mapping study on combinatorial interaction testing for software product lines. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, pp. 1–10.

Malhotra, R., 2015. A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. 27, 504–518.

Martinez, J., Sottet, J.-S., Frey, A.G., Bissyandé, T.F., Ziadi, T., Klein, J., Temple, P., Acher, M., Le Traon, Y., 2018. Towards estimating and predicting user perception on software product variants. In: International Conference on Software Reuse. Springer, pp. 23–40.

Mathur, A.P., 2008. Foundations of Software Testing. Pearson Education, India.

Medeiros, F., Kästner, C., Ribeiro, M., Gheyi, R., Apel, S., 2016. A comparison of 10 sampling algorithms for configurable systems. In: Proceedings of the 38th International Conference on Software Engineering. ACM, pp. 643–654.

Mendonca, M., Wasowski, A., Czarnecki, K., Cowan, D., 2008. Efficient compilation techniques for large scale feature models. In: Proceedings of the 7th International Conference on Generative Programming and Component Engineering. ACM, pp. 13–22.

Molnar, C., 2019. Interpretable machine learning. https://christophm.github.io/interpretable-ml-book/.

Morin, B., Barais, O., Jézéquel, J., Fleurey, F., Solberg, A., 2009. Models@ run.time to support dynamic adaptation. IEEE Comput. 42 (10), 44–51. http://dx.doi.org/10.1109/MC.2009.327.

Murashkin, A., Antkiewicz, M., Rayside, D., Czarnecki, K., 2013. Visualization and exploration of optimal variants in product line engineering. In: Proceedings of the 17th International Software Product Line Conference. ACM, pp. 111–115.

Murwantara, I.M., Bordbar, B., Minku, L.L., 2014. Measuring energy consumption for web service product configuration. In: Proceedings of the 16th International Conference on Information Integration and Web-Based Applications &#38; Services, IiWAS '14. ACM, New York, NY, USA, pp. 224–228. http://dx.doi.org/10.1145/2684200.2684314, URL http://doi.acm.org/10.1145/2684200.2684314.

Nair, V., Menzies, T., Siegmund, N., Apel, S., 2017. Using bad learners to find good configurations. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4–8, 2017. pp. 257–267. http://dx.doi.org/10.1145/3106237.3106238, URL http://doi.acm.org/10.1145/3106237.3106238.

Nair, V., Menzies, T., Siegmund, N., Apel, S., 2018. Faster discovery of faster system configurations with spectral learning. Autom. Softw. Eng. 1–31.

Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S., 2018. Finding faster configurations using flash. IEEE Trans. Softw. Eng.

Ochoa, L., Gonzalez-Rojas, O., Juliana, A.P., Castro, H., Saake, G., 2018. A systematic literature review on the semi-automatic configuration of extended product lines. J. Syst. Softw. 144, 511–532.

Ochoa, L., González-Rojas, O., Thüm, T., 2015. Using decision rules for solving conflicts in extended feature models. In: International Conference on Software Language Engineering (SLE). ACM, pp. 149–160.

Oh, J., Batory, D.S., Myers, M., Siegmund, N., 2017. Finding near-optimal configurations in product lines by random sampling. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4–8, 2017. pp. 61–71. http://dx.doi.org/10.1145/3106237.3106273, URL http://doi.acm.org/10.1145/3106237.3106273.

Osogami, T., Kato, S., 2007. Optimizing system configurations quickly by guessing at the performance. In: ACM SIGMETRICS Performance Evaluation Review, Vol. 35. ACM, pp. 145–156.

Pereira, J.A., Acher, M., Martin, H., Jézéquel, J.-M., 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. URL https://hal.inria.fr/hal-02356290.

Pereira, J.A., Constantino, K., Figueiredo, E., 2015. A systematic literature review of software product line management tools. In: International Conference on Software Reuse (ICSR). Springer, pp. 73–89.

Pereira, J.A., M., Acher, H., Martin, J., Jézéquel, G., Botterweck, A., Ventresque, 2019. Learning software configuration spaces: A systematic literature review. URL https://github.com/VaryVary/ML-configurable-SLR. (Accessed 04 June 2019).

Pereira, J.A., Matuszyk, P., Krieter, S., Spiliopoulou, M., Saake, G., 2018. Personalized recommender systems for product-line configuration processes. Comput. Lang. Syst. Struct..

Pereira, J.A., Schulze, S., Figueiredo, E., Saake, G., 2018. N-dimensional tensor factorization for self-configuration of software product lines at runtime. In: Proceeedings of the 22nd International Conference on Systems and Software Product Line-Volume 1. ACM, pp. 87–97.

Plazar, Q., Acher, M., Perrouin, G., Devroey, X., Cordy, M., 2019. Uniform sampling of sat solutions for configurable systems: Are we there yet? In: ICST 2019-12th International Conference on Software Testing, Verification, and Validation, Xian, China. pp. 1–12, URL https://hal.inria.fr/hal-01991857.

Pohl, K., Böckle, G., van der Linden, F.J., 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin Heidelberg.

Pohl, R., Lauenroth, K., Pohl, K., 2011. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE Computer Society, pp. 313–322.

Porter, A., Yilmaz, C., Memon, A.M., Schmidt, D.C., Natarajan, B., 2007. Skoll: A process and infrastructure for distributed continuous quality assurance. IEEE Trans. Softw. Eng. 33 (8), 510–525.

Putri, S.A., et al., 2017. Combining integreted sampling technique with feature selection for software defect prediction. In: 2017 5th International Conference on Cyber and IT Service Management (CITSM). IEEE, pp. 1–6.

Queiroz, R., Berger, T., Czarnecki, K., 2016. Towards predicting feature defects in software product lines. In: Proceedings of the 7th International Workshop on Feature-Oriented Software Development. ACM, pp. 58–62.

Roos-Frantz, F., Benavides, D., Ruiz-Cortés, A., Heuer, A., Lauenroth, K., 2012. Quality-aware analysis in product line engineering with the orthogonal variability model. Softw. Qual. J. 20 (3–4), 519–565.

Safdar, S.A., Lu, H., Yue, T., Ali, S., 2017. Mining cross product line rules with multi-objective search and machine learning. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, pp. 1319–1326.

Saleem, M.S., Ding, C., Liu, X., Chi, C.-H., 2015. Personalized decision-strategy based web service selection using a learning-to-rank algorithm. IEEE Trans. Serv. Comput. 8 (5), 727–739.

Samreen, F., Elkhatib, Y., Rowe, M., Blair, G.S., 2016. Daleel: Simplifying cloud instance selection using machine learning. In: NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium. IEEE, pp. 557–563.

Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K., 2015. Cost-efficient sampling for performance prediction of configurable systems (t). In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 342–352.

Sayagh, M., Kerzazi, N., Adams, B., Petrillo, F., 2018. Software configuration engineering in practice: Interviews, survey, and systematic literature review. IEEE Trans. Softw. Eng.

Sayyad, A.S., Menzies, T., Ammar, H., 2013. On the value of user preferences in search-based software engineering: a case study in software product lines. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, pp. 492–501.

Sharifloo, A.M., Metzger, A., Quinton, C., Baresi, L., Pohl, K., 2016. Learning and evolution in dynamic software product lines. In: 2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE, pp. 158–164.

Siegmund, N., Grebhahn, A., Apel, S., Kästner, C., 2015. Performance-influence models for highly configurable systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 284–294.

Siegmund, N., Kolesnikov, S.S., Kästner, C., Apel, S., Batory, D.S., Rosenmüller, M., Saake, G., 2012a. Predicting performance via automated feature-interaction detection. In: ICSE, pp. 167–177.

Siegmund, N., von Rhein, A., Apel, S., 2013a. Family-based performance measurement. In: ACM SIGPLAN Notices, Vol. 49. ACM, pp. 95–104.

Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P.G., Apel, S., Kolesnikov, S.S., 2011. Scalable prediction of non-functional properties in software product lines. In: Software Product Line Conference (SPLC), 2011 15th International, pp. 160–169.

Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P.G., Apel, S., Kolesnikov, S.S., 2013b. Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. Inf. Softw. Technol. 55 (3), 491–507.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G., 2012b. Spl conqueror: Toward optimization of non-functional properties in software product lines. Softw. Qual. J. 20 (3–4), 487–517.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Saake, G., 2008. Measuring non-functional properties in software product line for product derivation. In: 2008 15th Asia-Pacific Software Engineering Conference. IEEE, pp. 187–194.

Siegmund, N., Sobernig, S., Apel, S., 2017. Attributed variability models: outside the comfort zone. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, pp. 268–278.

Sincero, J., Schroder-Preikschat, W., Spinczyk, O., 2010. Approaching non-functional properties of software product lines: Learning from products. In: Software Engineering Conference (APSEC), 2010 17th Asia Pacific. pp. 147–155.

Song, C., Porter, A., Foster, J.S., 2013. itree: efficiently discovering high-coverage configurations using interaction trees. IEEE Trans. Softw. Eng. 40 (3), 251–265.

Stuckman, J., Walden, J., Scandariato, R., 2017. The effect of dimensionality reduction on software vulnerability prediction models. IEEE Trans. Reliab. 66 (1), 17–37.

Svahnberg, M., van Gurp, J., Bosch, J., 2005. A taxonomy of variability realization techniques: Research articles. Softw. Pract. Exper. 35 (8), 705–754. http://dx.doi.org/10.1002/spe.v35:8.

Temple, P., Acher, M., Biggio, B., Jézéquel, J.-M., Roli, F.,

Temple, P., Acher, M., Jézéquel, J., Barais, O., 2017a. Learning contextual-variability models. IEEE Softw. 34 (6), 64–70. http://dx.doi.org/10.1109/MS.2017.4121211, URL https://doi.org/10.1109/MS.2017.4121211.

Temple, P., Acher, M., Jézéquel, J.-M.A., Noel-Baron, L.A., Galindo, J.A., 2017b. Learning-Based Performance Specialization of Configurable Systems. Research report, IRISA, Inria Rennes; University of Rennes 1, URL https://hal.archives-ouvertes.fr/hal-01467299.

Temple, P., Galindo Duarte, J.A., Acher, M., Jézéquel, J.-M., 2016. Using machine learning to infer constraints for product lines. In: Software Product Line Conference (SPLC), Beijing, China. http://dx.doi.org/10.1145/2934466.2934472, URL https://hal.inria.fr/hal-01323446.

Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K., 2013. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 847–855.

Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G., 2014. A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. 47 (1), 6.

Trubiani, C., Apel, S., 2019. Plus: Performance learning for uncertainty of software. In: International Conference on Software Engineering NIER. ACM.

Valov, P., Guo, J., Czarnecki, K., 2015. Empirical comparison of regression methods for variability-aware performance prediction. In: SPLC'15.

Valov, P., Petkovich, J.-C., Guo, J., Fischmeister, S., Czarnecki, K., 2017. Transferring performance prediction models across different hardware platforms. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering. ACM, pp. 39–50.

Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B., 2017a. Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data. ACM, pp. 1009–1024.

Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B., 2017b. Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17. Association for Computing Machinery, New York, NY, USA, pp. 1009–1024. http://dx.doi.org/10.1145/3035918.3064029.

Varshosaz, M., Al-Hajjaji, M., Thüm, T., Runge, T., Mousavi, M.R., Schaefer, I., 2018. A classification of product sampling for software product lines. In: Proceeedings of the 22nd International Conference on Systems and Software Product Line-Volume 1. ACM, pp. 1–13.

Venkata, S.K., Ahn, I., Jeon, D., Gupta, A., Louie, C., Garcia, S., Belongie, S., Taylor, M.B., 2009. Sd-vbs: The san diego vision benchmark suite. In: 2009 IEEE International Symposium on Workload Characterization (IISWC). IEEE, pp. 55–64.

Švogor, I., Crnković, I., Vrček, N., 2019. An extensible framework for software configuration optimization on heterogeneous computing systems: Time and energy case study. Inf. Softw. Technol. 105, 30–42.

Weckesser, M., Kluge, R., Pfannemüller, M., Matthé, M., Schürr, A., Becker, C., 2018. Optimal reconfiguration of dynamic software product lines based on performance-influence models. In: Proceeedings of the 22nd International Conference on Systems and Software Product Line-Volume 1. ACM, pp. 98–109.

Westermann, D., Happe, J., Krebs, R., Farahbod, R., 2012. Automated inference of goal-oriented performance prediction functions. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). ACM, pp. 190–199.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, p. 38.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A., 2000. Experimentation in Software Engineering: An Introduction.

Xi, B., Liu, Z., Raghavachari, M., Xia, C.H., Zhang, L., 2012. Automated inference of goal-oriented performance prediction functions. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). ACM, pp. 190–199.

Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K., 2008. Satzilla: portfolio-based algorithm selection for sat. J. Artificial Intelligence Res. 32, 565–606.

Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., Talwadker, R., 2015. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015, pp. 307–319. http://dx.doi.org/10.1145/2786805.2786852. URL http://doi.acm.org/10.1145/2786805.2786852.

Yildiz, B., Bilbao, J.I., Sproul, A.B., 2017. A review and analysis of regression and machine learning models on commercial building electricity load forecasting. Renew. Sustain. Energy Rev. 73, 1104–1122.

Yilmaz, C., Cohen, M.B., Porter, A.A., 2006. Covering arrays for efficient fault characterization in complex configuration spaces. IEEE Trans. Softw. Eng. 32 (1), 20–34.

Yilmaz, C., Dumlu, E., Cohen, M.B., Porter, A., 2014. Reducing masking effects in combinatorialinteraction testing: A feedback drivenadaptive approach. IEEE Trans. Softw. Eng. 40 (1), 43–66.

Zhang, Y., Guo, J., Blais, E., Czarnecki, K., 2015. Performance prediction of configurable software systems by Fourier learning (t). In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 365–373.

Zhang, Y., Guo, J., Blais, E., Czarnecki, K., Yu, H., 2016. A mathematical model of performance-relevant feature interactions. In: Proceedings of the 20th International Systems and Software Product Line Conference. ACM, pp. 25–34.

Zheng, W., Bianchini, R., Nguyen, T.D., 2007. Automatic configuration of internet services. Oper. Syst. Rev. 41 (3), 219–229.

Zuluaga, M., Krause, A., Püschel, M., 2016. ε-pal: an active learning approach to the multi-objective optimization problem. J. Mach. Learn. Res. 17 (1), 3619–3650.

Juliana Alves Pereira is currently a Post-Doctoral researcher at PUC-Rio (Brazil). She was a researcher at the University of Rennes I (Inria/Irisa, France). Juliana received her Ph.D. degree with distinction in 2018 from the University of Magdeburg, Germany. Her research thrives to automate software engineering by combining methods from software analysis, machine learning, and meta-heuristic optimization. In recent years, she has published and revised research papers in premier software engineering conferences, symposiums, and journals. She is regularly presenting courses, tutorials, tools, and scientific results at national and international venues.

Mathieu Acher is Associate Professor at University of Rennes 1/Inria, France. His research focuses on reverse engineering, modeling, and learning variability of software intensive systems with different contributions published at ASE, ESEC/FSE, SPLC, MODELS, IJCAI, or JSS, ESEM journal. He was PC co-chair of SPLC 2017 and will be PC co-chair of VaMoS 2020. He is currently leading a research project on machine learning and variability.[12]

Hugo Martin is a Ph.D. Student at the University of Rennes 1, France. His research focuses on using interpretable machine learning to better understand configurable systems.

Dr. Jean-Marc Jzquel is a Professor at the University of Rennes and Director of IRISA, one of the largest public research lab in Informatics in France. He is also head of research of the French Cyber-defense Excellence Cluster and the director of the Rennes Node of EIT Digital. In 2016, he received the Silver Medal from CNRS. His interests include model driven software engineering for software product lines, and specifically component based, dynamically adaptable systems with quality of service constraints, including security, reliability, performance, timeliness, etc. He is the author of 4 books and more than 250 publications in international journals and conferences. He was a member of the steering committees of the AOSD and MODELS conference series. He also served on the editorial boards of IEEE Computer, IEEE Transactions on Software Engineering, the Journal on Software and Systems, the Journal on Software and System Modeling and the Journal of Object Technology. He received an engineering degree from Telecom Bretagne in 1986, and a Ph.D. degree in Computer Science from the University of Rennes, France, in 1989.

Goetz Botterweck is a Assoc. Professor in Computer Science at Trinity College, Dublin Ireland and with Lero – the Irish Software Research Centre. Previously, he held positions at the University of Limerick, Ireland, as Lecturer in Computer Science and Senior Research Fellow. His research interests are model-driven software engineering, software evolution, and software product lines. Botterweck received a Ph.D. in computer science from the University of Koblenz. He was PC co-chair of SPLC 2015 and ICSR 2017.

Anthony Ventresque received his Ph.D. degree in Computer Science from the University of Nantes & INRIA France in 2008. Dr Ventresque is currently an Assistant Prof. in the School of Computer Science at University College Dublin, Ireland, and a Funded Investigator with Lero, the SFI Irish Software Research Centre. Previously, he held positions as Research Fellow at NTU, Singapore (20102011), UCD, Ireland (20122014), and IBM Research Dublin, Ireland (20142015).

[12] https://varyvary.github.io/.