Effect of Feature Subset Selection on Samplings for Performance Prediction of Configurable Systems

João Marcello Bessa Rodrigues[1], **Juliana Alves Pereira**[1]

[1]Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro – RJ – Brazil

jrodrigues@inf.puc-rio.br, juliana@inf.puc-rio.br

Abstract. *Organizations require personalized solutions to effectively address users' needs, and stay competitive in the market. In this context, configurable systems offer numerous configuration options to meet user-specific functional and non-functional requirements. However, although configurability makes these systems flexible and versatile, a simple change can result in serious problems in different software variants, such as performance bottlenecks and security issues. Thus, automated approaches based on machine learning have been developed to facilitate configuration management. Our work aims to expand upon previous findings in this field by assessing their applicability to other scenarios. By introducing more efficient practices, we can contribute to cost reduction, higher software quality, and quicker time-to-market. This is particularly relevant in a global context where software plays a crucial role.*

1. Introduction

Information systems are highly configurable and offer a wide variety of features. Features determine the characteristics and functionality of the system [Teaff et al. 2019, Heradio et al. 2022]. They can be adjusted to meet the specific needs of users' functional and non-functional requirements (NFRs). Functional requirements define what the information system should do and how it should respond to different inputs and events. NFRs refer to the performance, quality, and operational characteristics of the system. They are related to execution time, memory consumption, security, and many other aspects. The system's configurability allows users to customize features to align with their unique needs and constraints, such as optimizing execution time, reducing memory consumption, or enhancing security measures. However, finding the appropriate customization can be a complex task in configurable systems. Configurable systems often offer numerous features, leading to an overload of options. Poor configuration decisions can result in various issues, such as, system failures, service interruptions, financial losses, and performance bottlenecks. This challenge persists even for experienced users with a deep understanding of the system's functionality.

To support users in making informed decisions, automated software approaches based on machine learning (ML) have been proposed in the literature [Pereira et al. 2021]. These approaches help users understand the implications of their choices and how they will impact the system's NFRs. They follow a *"sampling, measuring, learning"* process [Pereira et al. 2021]. The sampling phase aims to efficiently select a small and representative sample set of configurations from the vast space of all possible configurations.

The measuring phase involves executing the system for the sample set of selected configurations and measuring the relevant NFRs. The learning phase leverages the data collected in the measuring phase to train a model and identify patterns between features. For example, the model may point out that changing one configuration option may affect the performance or security of other parts of the system, thus assisting users in understanding the potential trade-offs associated with different configuration choices.

One challenging issue in this area is the time required for training ML models. Recently, Acher *et al.*[Acher et al. 2022] proposed a new approach for selecting a small and representative subset of features to train a predictive model and thus save time. This approach is then compared with a set of baseline approaches that use the complete set of features. The authors found that a very small set of features is need to obtain a highly accurate model. Furthermore, their study demonstrated the notable superiority of the proposed approach over other approaches when it comes to training time. The authors focused solely on the Linux operating system as a case study. In addition, it exclusively used the random sampling strategy for gathering data and conducting experiments.

In this paper, we conduct a replication of the [Acher et al. 2022] study to investigate the effect of their proposed approach over different systems and sampling strategies. The goal of our experiments is to determine whether the approach proposed in [Acher et al. 2022] over a particular subject system is as effective across *different* application scenarios and domains. Several sampling strategies have been devised in recent years [Pereira et al. 2021]. Our analysis across different sampling and application scenarios is inspired by the research conducted by Kaltenecker *et al.* [Kaltenecker et al. 2019] and Pereira *et al.* [Alves Pereira et al. 2020]. Their study emphasizes the importance of carefully selecting sampling strategies to ensure the accuracy of the prediction model.

However, these studies did not investigate whether sampling strategies are generalized across different algorithms. They use a single linear regression ML algorithm. By replicating and expanding on these findings, we intend to offer a more comprehensive understanding of how the feature selection algorithm proposed in [Acher et al. 2022] adapts to the sampling strategies in *et al.* [Kaltenecker et al. 2019, Alves Pereira et al. 2020]. In this context, we investigate to what extent learning approaches are sensitive to different configurable systems and different performance properties: Is there a dominant sampling strategy and learning approach that practitioners can always rely on? We hypothesize that practitioners can rely on a one-size-fits-all sampling strategy and learning approach that is cost-effective whatever the factors influencing the distribution of different configuration spaces. On the contrary, another hypothesis is that practitioners should change their sampling strategy and learning approach each time a system or a performance property is targeted. To test our hypothesis, we systematically report the influence of sampling strategies and learning approaches on the accuracy of performance predictions over the number of features and configurations used to train a model.

We make the following five contributions:

1. The use of different systems is not explored in [Acher et al. 2022]. Thus, we gather preliminary insights about the effectiveness of their proposal approach across eight systems.
2. The use of different learning algorithms and parameter tuning are not explored in [Kaltenecker et al. 2019, Alves Pereira et al. 2020]. These strategies may have a

strong influence on the prediction accuracy of different sampling strategies. We gather preliminary insights about the effectiveness of eight learning algorithms.

3. The use of different systems and sampling strategies is not explored in [Acher et al. 2022]. Thus, we gather preliminary insights about the effectiveness of six sampling strategies on eight systems and six performance properties.
4. A comparison of a wide range of ML algorithms and the effects of tree-based feature selection and sampling strategies over training time.
5. We have made all the artifacts publicly available at [art 2023].

2. Design Study

We follow four learning phases: *"sampling, measurement, learning, and evaluation"*. We replicate the sampling studies of [Kaltenecker et al. 2019] and [Alves Pereira et al. 2020]. We build and measure the performance property of each generated sampled configuration. The sample was partitioned into training and testing sets (10%, 20%, ..., 90%). Through *feature construction*, we develop attributes correlated with the target to boost the model's effectiveness [Vouk et al. 2023]. In the *preprocessing* phase, we address feature collinearity by removing mutually exclusive features and refining the model input for better accuracy and efficiency. *Feature selection* employs tree-based algorithms like Random Forest to prioritize features, enhancing training with top-ranked features (see Section2.3). This method was evaluated against a baseline that uses the complete set of features and random sampling. Next, we introduce our research questions, the subject systems used, and the experiment setup.

2.1. Research Questions

To identify relevant systems beyond the Linux kernel, we relied on a systematic literature review conducted by [Pereira et al. 2021]. Based on that work, we have selected a set of eight configurable systems that are suitable for our experiments. We aim at answering the following three research questions:

RQ1. How accurate is the prediction model with and without feature selection? To answer this question, we will compare the accuracy of the model using a subset of relevant features as input against the use of the complete set of features as input. Accuracy was measured using the Mean Absolute Percentage Error (MAPE).

RQ2. To what extent does the use of different sampling strategies degrade the accuracy of a prediction model with and without feature selection? The main goal of this question is to assess the effectiveness of various sampling strategies derived from a previous study in the field [Kaltenecker et al. 2019, Alves Pereira et al. 2020]. While the previous studies consider all features to make predictions, the objective of this study is to analyze the impact of different sampling approaches on the accuracy of performance predictions with a restricted set of features taken as input.

RQ3. How many computational resources are saved with feature selection? To assess computational resources saved with feature selection, we measure the time required to train each used learning model and sampling strategy. The training time was compared between models using the complete set of features and the subset of relevant features.

2.2. Subject System and Non-Functional Properties

To select subject systems for our study, we employed criteria based on dataset availability and verifiability from 69 systematic literature review studies [Pereira et al. 2021]. Initially, 46 studies were excluded due to inaccessible links or missing dataset repositories. We further disqualified 13 studies lacking an available feature model, crucial for sample generation as per [Kaltenecker et al. 2019]. This filtration left us with eight systems after removing two for inadequate dataset details (refer to Table 1). Our selection spans various domains, feature and configuration counts, and performance metrics, ensuring a comprehensive analysis framework.

Table 1. Table 1. Overview of Information Systems: Domain, Valid Configurations (#C), Features (#O), and Performance Metric.

Name	Domain	#C	#F	Performance
7Z	File archive utility	68,640	45	Compression time
BDB-C	Embedded database	2,560	20	Response time
DUNE	Multigrid solver	2,304	59	Solving time
HIPAcc	Image processing	13,485	56	Solving time
LLVM	Compiler	1,024	13	Compilation time
LRZIP	File archive utility	432	21	Compression time
POLLY	Code optimizer	60,000	41	Runtime
X264	Video encoder	1,152	18	Encoding time

2.3. Selection of Machine Learning Algorithms

In replicating the original study [Acher et al. 2022], we used the same list of ML algorithms, focusing on eight, including Linear Regression [Jamshidi et al. 2019], Ridge Regression [Rajan 2022, Chang et al. 2017], and others such as Random Forest [Amand et al. 2019, Bao et al. 2018], which is central to our methodology for feature selection. Our goal is to efficiently select essential features for machine learning models to balance resource usage and accuracy.

2.4. Selection of Sampling Strategies

In replicating the studies of [Acher et al. 2022] and [Kaltenecker et al. 2019], we used a list of six sampling strategies: Random Sampling [Guo et al. 2013], Solver-based [Pereira et al. 2021], Rand Solver-based [Chen et al. 2005, Chen et al. 2004], Coverage-based [Johansen et al. 2012, Marijan et al. 2013], Distance-based [Kaltenecker et al. 2019], and Div Distance-based [Kaltenecker et al. 2019]

2.5. Experiment Setup

To assess the accuracy of ML models for each system, the authors partition the dataset into training and test subsets. This division is performed using the cross-validation technique, where the dataset is randomly split into k subsets. Each subset is used as a test set once, while the remaining k-1 subsets are used to train the model. Cross-validation allows for evaluating the generalization and robustness of the trained models, ensuring that the results are not influenced by a single random data split. To evaluate the accuracy of our models, we employed the Mean Absolute Percentage Error (MAPE) as our primary metric. We used MAPE to allow us to compare our results with the results of [Acher et al. 2019].

This study considers different training scenarios by varying the training and testing sizes (10%, 20%, 30%, ..., 90%) and the number of examples used in creating the feature ranking lists (30%, 40%,50%, 60%). We analyze the MAPE for each setup, both with and without feature selection, extending Acher *et al.*'s study (Section 2.1 – *RQ1*). Additionally, we examine the impact of different sampling strategies on model accuracy (Section 2.1 – *RQ2*). Finally, we measured the computational resources required for training our models. We focus on evaluating any potential savings achieved through feature selection (Section 2.1 – *RQ3*). Then, we repeated the process multiple times and reported the average performance MAPE, to ensure consistent and reliable results.

2.6. Artifact Availability

The artifacts of our study are available in our supplementary material [art 2023]. Our material contains a notebook that is an efficient resource for exploring the available configurable system measurements. This notebook is thoughtfully structured into four distinct sections - *sample*, *measurement*, *learning*, and *evaluation*.

3. Results

We now present our results and answer the research questions defined in Section 2.

3.1. RQ1. Learning Accuracy with Feature Selection

Table 2 presents the average MAPE of the best statistical learning algorithms for each system, for various sizes of training set (N), both without and with tree-based feature selection. The sample was obtained using a random sampling strategy, as in the original study [Acher et al. 2022]. We show the best algorithm from a set of eight algorithms, which includes Linear Regression, Ridge Regression, Lasso Regression, Polynomial Regression, ElasticNet, Decision Tree, Random Forest, and GBTree.

Table 2. MAPE of the best learning algorithm for the prediction of performance for each system, without and with tree-based feature selection, with N being the percentage of the dataset used as training.

System	Without Feature Selection					With Feature Selection				
	Algorithm	N=30	N=50	N=70	N=90	Algorithm	N=30	N=50	N=70	N=90
7z	RandomForest	4.53	3.73	3.45	3.35	RandomForest	5.57	5.06	4.87	4.83
BDB-C	DecisionTree	5.98	1.01	0.45	0.58	DecisionTree	0.82	0.67	0.62	0.51
Dune	RandomForest	6.73	5.32	4.94	4.61	GBTree	9.34	8.71	8.15	8.30
Hipacc	RandomForest	3.59	2.18	1.66	1.54	RandomForest	4.32	3.68	3.64	3.57
Lrzip	DecisionTree	12.73	13.39	11.07	2.79	DecisionTree	28.31	29.89	24.44	17.93
LLVM	LinearRegression	2.65	2.88	3.18	2.88	GBTree	2.18	1.59	1.32	1.80
Polly	RandomForest	1.70	1.24	1.17	1.06	RandomForest	1.35	1.19	1.19	1.20
x264	DecisionTree	2.13	1.40	0.57	0.18	GBTree	1.17	1.04	0.44	0.38

To rank the algorithms, we calculated a weighted value of the MAPE, taking into account the training percentage. Detailed results for all eight algorithms in each system, using training rates from 10% to 90% and feature selection from 30% to 60%, are available in our supplementary material [art 2023]. Most algorithms met our set time and memory limits, except Polynomial Regression without feature selection, which did not scale to degree 2, similar to observations in the original study for the Linux Kernel.

3.1.1. What is the best algorithm to learn performance?

In our analysis, Random Forest, Decision Tree, and Linear Regression were identified as leading algorithms without feature selection, aligning with [Acher et al. 2022]'s findings, with an exception for Linear Regression. Random Forest excelled in systems with extensive feature sets, such as DUNE and HIPAcc (see Table 1). Most algorithms showed performance gains with increased training data, excluding BDB-C and LLVM, which peaked at 70% and 30% training data, respectively, suggesting feature set size impacts.

Minimal MAPE variance was observed between 50%–70% and 70%–90% training data increments across systems, with lrzip being an exception. Linear Regression and GB-Tree delivered comparable results for LLVM, with MAPEs between 2.86 and 3.26. High MAPE rates were noted in BDB-C and Lrzip across algorithms, barring Decision Tree and Random Forest, which notably excelled, especially in feature-rich systems with ample training data.

3.1.2. How many features do we need to learn performance?

Figure 1 demonstrates the impact of feature selection on system accuracy with a 70% training set. It specifically shows Random Forest's performance for feature selections between 30% and 60%, indicating stable accuracy, especially between 50% and 60%, where the mean absolute percentage error (MAPE) remains under 10%, except for lrzip at N=90%. These findings, summarized in Table 2 for a 50% feature selection scenario, confirm that a limited number of features significantly affect system performance.

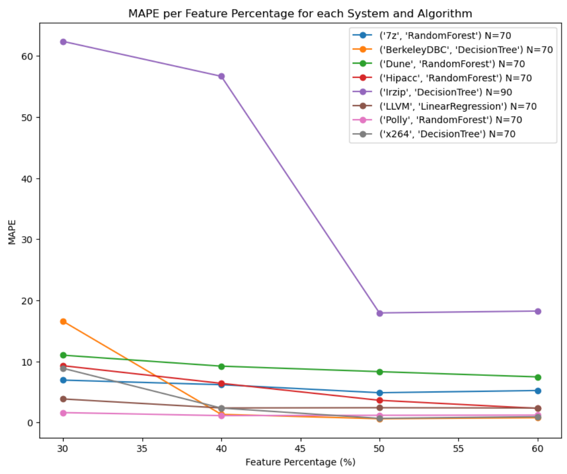


Figure 1. Value of MAPE for the percentage of selected features with the training set N=70% for the best algorithm mentioned in Table 2.

3.1.3. What is the effect of feature selection?

Table 2 reveals that models trained with 50% of top-ranked features exhibit diverse MAPE outcomes, with Tree-based algorithms (Random Forest, GBTree, Decision Tree) showing enhanced performance, aligning with [Acher et al. 2022]'s Linux Kernel analysis. Notably, larger training sets didn't consistently lower MAPE. Some systems experienced minor MAPE increases (up to 0.15%) with training data expansion from 70% to 90%, except for lrzip and LLVM, particularly in feature-rich systems (7z, Dune, Hipacc).

Feature selection's impact varied: it reduced average MAPE in feature-sparse systems (LLVM, BDB-C, x264) but was less beneficial in systems with over 30 features. Linear regression variants (Lasso, Ridge, Elasticnet) underperformed in both scenarios, underscoring the complexity of performance prediction. A notable observation was at 70% training data for LLVM, where MAPE spiked by 2.48 points (1.32% vs 3.18%) with tree-based selection, hinting at improved precision. MAPE stability from 70% to 90% training data was consistent, setting a 70% standard training set size for future inquiries, with lrzip as an exception at 90%.

3.2. RQ2. Sampling Strategies Accuracy

Table 3 compares the MAPE of different sampling strategies for models trained with and without feature selection (50% and 100% of features, respectively), using 70% of configurations for training. The best MAPEs, indicating the lowest error percentage for each system, are highlighted in bold, based on the best algorithms identified in RQ1.

Table 3. Systems best learning algorithm MAPE for sampling strategies using 50% or 100% of selected features with 70% of the dataset used as training.

		Distance-based		Div Distance-based		Rand Solver-based		Random Sampling		Solver-based		Coverage-based	
System	Algorithm	50	100	50	100	50	100	50	100	50	100	50	100
7z	RandomForest	**3.94**	**3.45**	4.21	4.09	5.51	5.09	4.87	5.28	4.07	3.89	4.07	3.98
DBC-C	DecisionTree	**0.53**	**0.45**	0.55	0.64	1.56	1.45	0.62	0.78	0.54	0.51	0.98	0.81
Dune	GBTree	**6.00**	**6.68**	7.02	7.33	9.76	9.28	8.15	7.36	8.02	7.78	8.98	7.63
	RandomForest	**5.49**	**4.94**	7.03	5.51	9.38	7.07	8.33	6.33	8.46	6.12	8.75	6.17
Hipacc	RandomForest	**3.25**	**1.66**	4.01	2.40	3.98	2.25	3.64	2.20	3.55	1.88	3.28	2.03
Lrzip	DecisionTree	36.10	11.07	52.82	3.52	56.05	2.99	24.44	3.45	**6.30**	**2.68**	43.82	4.29
LLVM	GBTree	2.93	2.86	1.89	2.21	**1.25**	**1.26**	1.32	1.50	2.14	2.19	1.81	1.86
	LinearRegression	3.15	3.18	2.83	2.97	**1.20**	**1.24**	2.42	2.48	2.51	2.45	2.19	2.44
Polly	RandomForest	1.18	1.17	1.29	1.31	**0.89**	1.67	1.19	1.15	1.06	**1.06**	1.28	1.33
x264	DecisionTree	1.10	0.57	1.45	1.30	1.13	**0.24**	0.68	1.01	**0.20**	0.27	0.76	0.70
	GBTree	0.82	0.81	0.75	0.83	0.65	0.30	0.44	0.84	**0.22**	**0.16**	0.57	0.48

Distance-based sampling excelled in four systems (7z, BerckleyDBC, Dune, and Hipacc) using tree-based learning algorithms. Hipacc had nearly twice the accuracy with feature selection. Solver-based sampling was more effective for Polly, Lrzip, and x264, with x264 showing lower error rates both with and without feature selection. We can also observe that Solver-based obtained the best values for two of the three systems where Decision Tree was the training algorithm with the highest accuracy, with Lrzip (with feature selection) having significantly better results in comparison to other sampling strategies. The *Rand Solver-based* strategy stood out in three systems (LLVM, Polly, and x264), with results below 1.3% MAPE for tree-based algorithms and linear regression. Distance-based, Rand Solver-based, and Solver-based sampling strategies result in the most accurate performance models for the eight subject systems and dataset sizes. We identified that Rand Solver-based sampling obtained the best MAPEs. Rand Solver-based also stood out for the best results on four distinct types of learning algorithms.

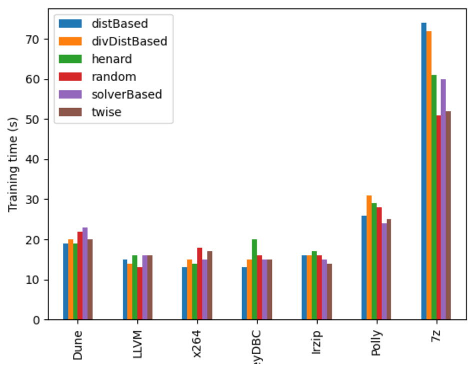
3.3. RQ3. Training Time

The computational resources used for training with and without feature selection were analyzed in a controlled environment[1], where all the processing time information was obtained for all systems, sample algorithms, and training sets. For training sets without

[1]We used an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, with 96GB RAM, and an SSD of 1TB.

feature selection, considering the total computational time for the six sampling strategies used, ranged from 45 to 4,091 seconds. Figure 2 (left) shows the computation times of these systems for each sampling algorithm, considering all system features and using 70% of the database for training. For Random Forest, we reported for x264 a 45-second computation over 19 features and 212 rows of the dataset. 7z took 546 seconds computational over 4,091 dataset rows for 46 features. BDB-C, Dune, Hipacc, Lrzip, LLVM, and Polly reported 79, 189, 486, 88, 60, and 234 seconds of computation time, respectively.

[100% of the selected characteristics.]



[50% of the selected characteristics.]

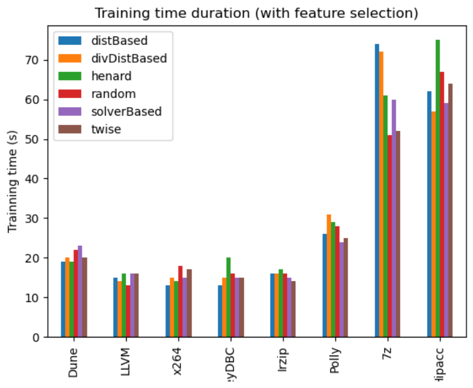


Figure 2. Algorithm training time considering 100% and 50% of the selected features with training set N=70% for the best algorithm mentioned in Table 2.

The graph in Figure 2 (right) presents the system improvement rate with feature selection. It also shows that for systems under 30 features, tree-based feature selection increased the training time from 8% to 105%. However, despite the highest rates of increase in training time, the smaller systems improved their accuracy by up to 200%, as can be seen with the LLVM system. The systems that achieved improvement in training time were not able to achieve rates as significant as those obtained in the original study for the Linux system. Dune, 7z, Polly, and Hipacc improved by 34%, 31%, 30%, and 20% respectively for 50% of the selected features, while the Linux system reached values above 1,600% for 5.7% of the selected features.

4. Discussion

This research extends the current understanding of ML approaches' effectiveness in system configuration, thus carrying a variety of practical implications for different stakeholders. It validates that for systems with more than 30 features, tree-based feature selection approaches can offer computational savings and improve model accuracy. Conversely, for smaller systems with fewer features, these approaches might extend the training period without significantly enhancing performance, suggesting that ML approach selection should be adapted to the system's specific characteristics.

Regarding sampling strategies, results vary significantly depending on the system, properties, and learning algorithms used. While distance-based sampling is shown to be more effective for medium to large sample sets, and random sampling is preferred for half of the systems, there is not a universally superior dominant strategy across all contexts. Coverage-based sampling presents promising results in terms of balancing sample size and prediction model accuracy. Additionally, the study also highlights a significant limitation in replicating sampling strategies in complex systems like the Linux kernel, due to the lack of available feature models. Future exploration of the use of transfer learning as a potential solution when feature models are not available is suggested.

By replicating the research across eight systems beyond Linux, it was found that the trends and methodologies observed could be applied to other configurable systems, although results may vary. The text emphasizes the importance of feature selection influenced by both sampling method analysis and consideration of static software metrics, suggesting a path for future investigations into the interplay between static software metrics and the feature selection process.

5. Conclusion

We investigate the advantages of using a subset of options when configuring information systems to obtain faster, simpler, and more accurate performance models. This study investigates six sampling strategies [Kaltenecker et al. 2019, Alves Pereira et al. 2020] to build upon the precedents established by successful implementations in complex systems, like the Linux kernel [Acher et al. 2022]. Our work has made a substantial contribution by offering research artifacts that improve the dependability and reproducibility of ML applications in configurable systems.

In the context of Information Systems (IS), our findings hold substantial relevance. The approach offers a strong foundation for incorporating innovative ML approaches into the system development lifecycle, which promotes and raises the level of software solutions for IS practitioners and developers. It directly contributes to the operational efficiency and effectiveness of IS through the simplification of configuration management processes. Thus, reducing computational resources and facilitates a more intuitive understanding of configuration impacts on performance.

References

(2023). Effect of feature subset selection on samplings (artifact). `https://anonymous.4open.science/r/FSE2024-785D/`. Accessed: 2023-09-28.

Acher, M., Martin, H., Lesoil, L., Blouin, A., Jézéquel, J.-M., Khelladi, D. E., Barais, O., and Pereira, J. A. (2022). Feature subset selection for learning huge configuration spaces: the case of linux kernel size. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A*, pages 85–96.

Acher, M., Martin, H., Pereira, J. A., Blouin, A., Jézéquel, J.-M., Khelladi, D. E., Lesoil, L., and Barais, O. (2019). *Learning very large configuration spaces: What matters for linux kernel sizes*. PhD thesis, Inria Rennes-Bretagne Atlantique.

Alves Pereira, J., Acher, M., Martin, H., and Jézéquel, J.-M. (2020). Sampling effect on performance prediction of configurable systems: A case study. page 277–288, New York, NY, USA. Association for Computing Machinery.

Amand, B., Cordy, M., Heymans, P., Acher, M., Temple, P., and 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*, page 7. ACM.

Bao, L., Liu, X., Xu, Z., and Fang, B. (2018). Autoconfig: automatic configuration tuning for distributed message systems. pages 29–40.

Chang, X., Lin, S.-B., and Zhou, D.-X. (2017). Distributed semi-supervised learning with kernel ridge regression. *The Journal of Machine Learning Research*, 18(1):1493–1514.

Chen, T. Y., Leung, H., and Mak, I. K. (2005). Adaptive random testing. In Maher, M. J., editor, *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, pages 320–329, Berlin, Heidelberg. Springer Berlin Heidelberg.

Chen, T. Y., Merkel, R., Wong, P., and Eddy, G. (2004). Adaptive random testing through dynamic partitioning. In *Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings.*, pages 79–86. IEEE.

Guo, J., Czarnecki, K., Apel, S., Siegmund, N., and Wasowski, A. (2013). Variability-aware performance prediction: A statistical learning approach. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 301–311. IEEE.

Heradio, R., Fernandez-Amoros, D., Galindo, J. A., Benavides, D., and Batory, D. (2022). Uniform and scalable sampling of highly configurable systems. *Empirical Softw. Engg.*, 27(2).

Jamshidi, P., Cámara, J., Schmerl, B., Kästner, C., and Garlan, D. (2019). Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. *arXiv preprint arXiv:1903.03920*.

Johansen, M. F., Haugen, Ø., and Fleurey, F. (2012). An algorithm for generating t-wise covering arrays from large feature models. In *SPLC'12*, pages 46–55.

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

Marijan, D., Gotlieb, A., Sen, S., and Hervieu, A. (2013). Practical pairwise testing for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, page 227–235, New York, NY, USA. Association for Computing Machinery.

Pereira, J. A., Acher, M., Martin, H., Jézéquel, J.-M., Botterweck, G., and Ventresque, A. (2021). Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software*, 182:111044.

Rajan, M. (2022). An efficient ridge regression algorithm with parameter estimation for data analysis in machine learning. *SN Computer Science*, 3(2):171.

Teaff, J., Young, B., and Clements, P. (2019). Applying feature-based systems and software product line engineering in unclassified and classified environments. *INCOSE International Symposium*, 29:269–283.

Vouk, B., Guid, M., and Robnik-Šikonja, M. (2023). Feature construction using explanations of individual predictions. *Engineering Applications of Artificial Intelligence*, 120:105823.