Out-of-the-Box Prediction of Non-Functional Variant Properties Using Automated Machine Learning

Lukas Güthing
lukas.guething@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Tobias Pett
tobias.pett@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Ina Schaefer
ina.schaefer@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

ABSTRACT

A configurable system is characterized by the configuration options present or absent in its variants. Selecting and deselecting those configuration options directly influences the functional properties of the system. Apart from functional properties, there are system characteristics that influence the performance (e.g., power demand), safety (e.g., fault probabilities), and security (e.g., susceptibility to attacks) of the system, called Non-Functional Properties (**NFPs**). Knowledge of NFPs is crucial for evaluating a system's feasibility, usability, and resource demands. Although variability influences these characteristics, NFPs do not compose linearly for every selected feature. Feature interactions can increase the overall NFP values through (potentially exponential) amplification or decrease them through mitigation effects. In this paper, we propose an automated machine learning (**AutoML**) approach to predict NFP values for new configurations based on previously measured configuration values. Using AutoML, we leverage the advantages of machine learning for predicting NFPs without having to parameterize and fine-tune machine learning models. This approach and the resulting pipeline aim to reduce the complexity of performance prediction for configurable systems. We test the feasibility of our pipeline in a first evaluation on 4 real-world subject systems and discuss cases where AutoML may improve the prediction of NFPs.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Software product lines, Cyber-physical systems, Machine learning, AutoML

ACM Reference Format:
Lukas Güthing, Tobias Pett, and Ina Schaefer. 2024. Out-of-the-Box Prediction of Non-Functional Variant Properties Using Automated Machine Learning. In *28th ACM International Systems and Software Product Line Conference (SPLC '24), September 02–06, 2024, Dommeldange, Luxembourg*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3646548.3676546

SPLC '24, September 02–06, 2024, Dommeldange, Luxembourg

ACM ISBN 979-8-4007-0593-9/24/09
https://doi.org/10.1145/3646548.3676546

1 INTRODUCTION

As cyber-physical systems become more complex, product-line engineering gains importance due to increased functionality demands and the need for customer-tailored solutions [1, 6, 12]. Product lines consist of *variants*, which differ in user-experienceable characteristics of the software, called *features*. Features can be considered *functional* aspects of the product that includes the respective features. In addition to functional properties, variants possess *non-functional properties* (NFPs) that usually result from feature choices. NFPs include, for example, the system's power usage or memory consumption, but also safety- and security-relevant characteristics like mean-time-to-failure or susceptibility to attacks. These characteristics are crucial information regarding the usability of a variant of the system. Information on a variant's power consumption or memory usage, for example, can determine the hardware requirements, or exclude existing hardware as feasible to run the variant's software. Meanwhile, safety and security metrics may constrain the system's applicability in domains with specifications and standards.

Measuring these NFPs for every variant is not feasible as the number of configurations and their respective derived variants increases rapidly for real-world systems. Additionally, feature-wise measurement, i.e., measuring the value of a single feature, is often impossible since an individual feature's NFP cannot be separated from other features present in the variant. Therefore, a process to predict NFP values for configurations that were not measured prior is necessary. However, predicting these characteristics is often difficult due to several aspects: First, they may be unknown for individual features. Predicting a variant's NFP value by composing the values of the present feature is therefore not possible. Second, NFPs are not strictly additive, as the interactions between features may influence NFP values linearly, exponentially, or even show mitigation effects. Thirdly, the influence on some NFPs may not be directly linked to certain features, but rather occur due to feature interactions. *Performance-influence models* have been developed to automatically derive the impact of feature interactions on NFP values for variants [9]. However, mathematical models can be inaccurate for prediction tasks and tools might not be able to detect non-trivial interactions and make them explicit.

Machine learning (ML) approaches have been developed to predict NFP values to tackle the complexity stemming from non-linear interactions of features [3]. As machine learning is mostly agnostic of the underlying variability model and potential feature interactions, these interactions do not have to be made explicit, saving effort in testing and measurement. Even though machine learning can perform well in prediction tasks, it often requires an in-depth understanding of the prediction problem and the machine learning

technique itself. The selection of the machine learning technique and parameter (fine-)tuning are important for the model's precision.

Automated Machine Learning (AutoML) is a concept that has been developed to simplify the usage of machine learning [11]. AutoML frameworks reduce the effort of selecting adequate machine-learning models and fine-tuning parameters by automating these tasks. Therefore, the user needs less understanding of machine learning concepts as long as the input for the models is correctly encoded. Therefore, automated machine learning increases the accessibility of machine learning and allows machine learning to be used more easily in other domains.

In this paper, we aim to leverage AutoML's easy-to-use characteristics to increase the accessibility of high-quality NFP prediction for SPLs. Machine learning has been shown to be proficient in prediction tasks for various problems, and AutoML removes the cumbersome tasks involved in using machine learning. Therefore, we propose an automated pipeline using the AutoML framework *AutoGluon* [2] which takes NFP measurements of variants as an input and predicts NFP values for previously unmeasured variants. We inspect the usefulness of the pipeline regarding prediction accuracy and run time. For the prediction accuracy, we pose the following research question: How accurate are NFP predictions from machine learning models trained by an automated machine learning framework (**RQ1**)? Regarding run time, we combine machine learning model training time and prediction time in one research question: How long does it take to predict NFPs using automated machine learning (**RQ2**)?

To summarize, our main contributions are:

- A pipeline leveraging automated machine learning (AutoML) to predict NFP values for configurations of configurable software based on the NFP values of previously measured configurations.
- A feasibility study on 4 real-world subject systems to show the accuracy and usability of the proposed pipeline.

2 STATE OF THE ART

In this section, we first recall relevant terms from (software) product line engineering. We review important milestones in the area of NFP prediction for configurable systems and machine learning automation. Based on these milestones, we identify the current research gap for automated NFP prediction. We then introduce automated machine learning and its fundamentals, which we later use to predict non-functional characteristics of variants to close this research gap. We explain all concepts using a simplified running example.

2.1 Product Lines

Product Line Engineering (PLE) aims to simplify the development of configurable systems. Product lines consist of different *variants*, i.e., products that differ in functionality but share the same domain. The differences between variants are defined by *features*. Features are user-experienceable characteristics usually realized in software and/or hardware artifacts. The selection and deselection of features for particular variants is represented by their respective *configurations*. In our running example, the features and valid configurations of a database system are defined by the feature model in Figure 1.

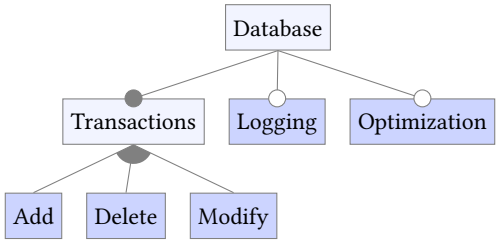

Figure 1: Feature model for the running example: A database with different transactions and optional logging and optimization.

ID	Add	Delete	Modify	Logging	Optimization	RAM
0	X	X				2 GB
1	X	X		X		4 GB
2	X	X			X	1 GB
3	X	X	X			3 GB
4	X	X		X	X	6 GB

Table 1: Running example: Configurations of a product line and their memory footprint.

Configurations also have non-functional properties that may result from the (de)selection of features. The non-functional properties range from ease of use, to memory and processor consumption, to safety-related properties such as error rates. The NFPs of a set of configurations of our running example are given in Table 1. Each line represents a configuration and each *X* in a line means that the column's feature is present in the respective configuration. However, the values of the NFPs are not trivially dependent on the selected features but can be influenced by certain combinations of the features present in a variant. *Feature interactions*, i.e., effects of two or more features interacting in potentially non-predicted ways, complicate the correct prediction of NFP values. Because of feature interactions, NFP values of features cannot simply be added to get the value for the whole variant [10]. In our running example (cf. Table 1), the *Logging* and *Transaction* features each add to the memory footprint of the variant in which they are present. *Optimization* reduces the memory footprint for *Transaction* features, however, the *Logging* is implemented so that it also logs the optimization steps. Therefore, the memory footprint of *Logging* increases with each selected *Transaction* feature when *Logging* and *Optimization* are selected simultaneously.

2.2 NFP Prediction for Product Lines

The information on the NFP values of variants is important in different stages of development and over the lifecycle of a configurable system. The trivial way to get the performance/memory footprint/etc. of a configurable system is to build all variants and measure the respective NFP values. This might be feasible for small software product lines but quickly becomes infeasible for systems with more configuration options based on an exponential increase of variants. Thus, NFP prediction is needed to support the development stages where information about NFP values is important.

There have been different approaches to predict non-functional properties for product-line variants: Siegmund et al. [8, 9] have developed approaches to identify feature interactions and derive performance-influence models, making the feature interactions that influence the NFP (i.e. performance) value explicit. Guo et al. [5] propose *DECART*, an extension to *CART* (Classification And Regression Trees) [4]. *DECART* systematically trains and tunes classification and regression trees to obtain a prediction model for the performance of variants. *DeepPERF*, proposed by Garg et al. [3], is a transformer model that suggests changes to source code to improve performance. The approaches mentioned above require an in-depth understanding of either the underlying variability model or the machine learning mechanisms used. Pereira et al. [7] give an extensive overview of the mentioned as well as further NFP prediction approaches.

In this work, we inspect the usefulness of automated machine learning for NFP prediction. In contrast to the presented approaches in the literature, we aim to eliminate the need for either analyses that make the impact of interactions on performance explicit, or for in-depth knowledge about machine-learning techniques.

2.3 AutoML

AutoML frameworks automatically train different machine-learning models, build ensembles of different models, and evaluate their performance to find the best fit for the respective problem instance, making it easier for end-users to use machine-learning concepts. In this work, we will use *AutoGluon* [2] as a regression predictor. This predictor is the core part of our pipeline for NFP prediction for product line configurations. The machine learning models used in *AutoGluon* can perform prediction tasks such as classification and regression. Additionally, *AutoGluon* outputs so-called *ensembles*, i.e., weighted compositions of different machine learning models that perform best as a combination instead of separately.

To our knowledge, AutoML has not yet been used for NFP prediction for configurable software. We expect AutoML to perform well due to the prediction potential of machine learning while requiring less knowledge about machine learning processes, therefore making the usage easier for end-users.

3 AUTOMATED MACHINE LEARNING PIPELINE

We present the pipeline that uses AutoML to predict NFP values for previously unmeasured variants in Figure 2. The pipeline starts by preparing training data for the machine learning models. We expect known configurations and NFP values collected for those configurations as training data. In the next step, we use *AutoGluon* to fit machine learning models to our input data. Finally, we use the trained model to predict NFP values for still-unknown configurations. In the following, we discuss each pipeline step in detail.

Step 1: Data Preparation

Machine learning algorithms usually expect their input data to be in specific formats. As we use AutoGluon's tabular data predictor in our pipeline, the input has to be tabular data. Therefore, we encode configurations and their NFP value in a tabular format. The output of the encoding is a CSV file with features as columns and the de-/selection of the respective feature a 0 or 1 in the respective line for each configuration. Additionally, the last column(s) of the data table is the NFP value for each configuration. The input file for our running example therefore looks similar to Table 1, except that the ID is not part of the file. The line for configuration *0* would then be `1 1 0 0 0 2`, where the first `5 0/1`s describe the (de-) selection of the database features and the 2 encodes the RAM footprint to be predicted later. Technically, there could be more than one predicted column with different NFP values. Since the input is not standardized, input data parsing has to be adapted to the respective input format. This data table then serves as input for the machine-learning model.

Step 2: AutoML Model Training

We choose *AutoGluon* [2] as an AutoML framework for our pipeline. *AutoGluon* preprocesses the tabular data and chooses machine learning models adequate to the type of problem, i.e., prediction or regression with binary or multiclass input. *AutoGluon* then also splits the input data into training and testing sets to train a set of different machine learning models. Once *AutoGluon* outputs a trained ensemble of models, this ensemble can then be used to predict NFP values. This ensemble is guaranteed to be the best performing — out of the ensembles trained in this run — on the training data.

Step 3: NFP Prediction

After Step 2, we use the predictor model ensemble *AutoGluon* outputs for NFP prediction. To predict NFPs for a new configuration, the input for the ensemble has to be the configuration in the same tabular format as the training input. For example, we might want to predict the memory footprint for the configuration with all features present in our running example. The configuration would then be encoded similarly to the training data, i.e., by `1 1 1 1 1`, meaning that all five features are selected. The last column is missing since that is the value to be predicted. *AutoGluon* takes the encoded configuration as input and outputs the NFP value the ensemble of machine learning models predicts.

4 FEASIBILITY STUDY

We run the proposed pipeline on data from real-world configurable systems and predict non-functional property values to evaluate it. This real-world data is taken from Siegmund et al. [10]'s work. We use four of their subject systems, for which real performance measurements are available, i.e., results from different benchmarks. As these subject systems have real measurements, we can evaluate the NFP prediction on real-world data. The dataset includes the systems shown in Table 2. We chose only a subset from the systems in Siegmund et al. [10]'s work as we wanted to inspect feasibility and not do an in-depth evaluation of the pipeline. Apache is a web server application, Berkeley is a database system (in two different implementations), and LLVM is a compiler system. We encoded the configurations and performance values in tabular data so that we can use it with *AutoGluon*. We answer **RQ1** (prediction accuracy) by splitting the set of configurations for each subject system into a training and a test set. We then train machine learning models with the training set and use the test set as ground truth to compare our predictions to. For **RQ2** (overall runtime), we measure the time

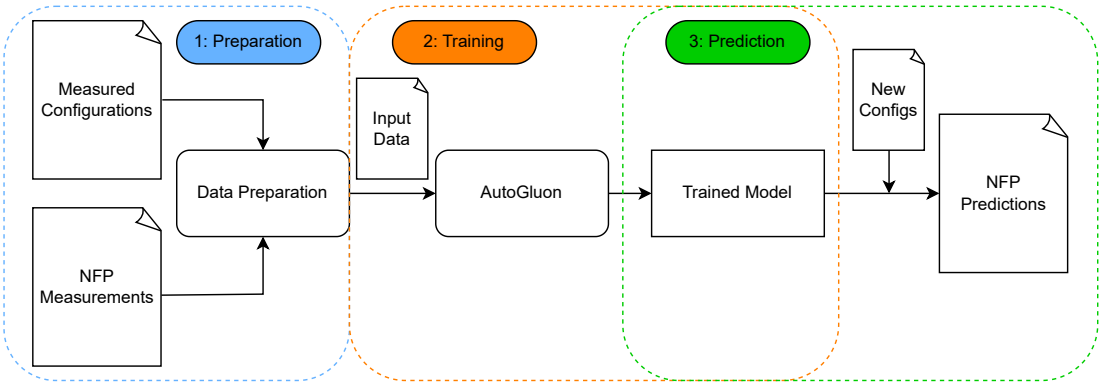

Figure 2: Pipeline for automated NFP prediction

Name	#Features	#Configurations
Apache	9	192
Berkeley (C)	18	2560
Berkeley (Java)	18	180
LLVM	11	1024

Table 2: Subject Systems

needed for the training by AutoGluon as well as for predicting NFP values for new configurations. The experiment was run on a MacBook Pro with an M3 chip and 16 GB of RAM. We did not run non-operating system software in parallel to reduce performance loss as much as possible.

RQ1: Prediction accuracy. We vary the size of the training sets because machine learning models can be trained with different amounts of information. Trivially, the prediction will most likely be accurate if we train a model with all of the configuration data. However, in the more realistic use case, information for a few configurations is available and the set of unmeasured configurations is much larger. Inspired by Guo et al. [5], we choose N to $4N$ configurations for training, with N being the number of features in the feature model of the respective system. To evaluate the accuracy, we (a) compare the prediction for configurations to their measured value and (b) calculate the mean absolute percentage error (MAPE). The MAPE is a common metric to compare predictions to actual values in statistics. The MAPE averages the relative error of all predictions respective to the actual value in the predicted cases. We use the two metrics (a) and (b) to gain an intuition into how the AutoML-based pipeline performs.

RQ2: Model Training and Prediciton Time. AutoGluon reports training times, the trained models and ensembles as well as the time needed for predictions. Therefore, we do not need to extend the pipeline further for time measurements. RQ2 helps understand the (computational) effort to train machine learning models and predict NFP values with the trained models.

Results and Discussion

RQ1: Prediction accuracy. The accuracy measurements scale in different dimensions: First, the accuracy correlates with the size of the training set, i.e., the accuracy improves with higher N. In Figure 3, every data point is one configuration, with the predicted value on the y-axis and the actual value on the x-axis. Points on the $y = x$ line are accurately predicted configurations. The second dimension in which accuracy varies is the subject systems. Figure 4 shows the prediction vs ground truth plots for all four subject systems with $4N$ configurations for training. The accuracy varies between the plots as well as the MAPE shown in Table 3.

Except for Apache $1N$ and $2N$, the MAPE decreases for higher N, i.e., the accuracy improves. While this was expected, the MAPE varies between the subject systems. The two different Berkeley implementations have a similar MAPE at $4N$, however, performance prediction for Java Berkeley performs relatively well for small N, while C Berkeley's prediction seems harder for the machine learning models. This difference is most likely due to the different number of valid configurations for the two systems. Additionally, there seem to be interactions in specific configurations that are either sparse in the configuration space or hard to learn for the machine learning models. For example, in Figure 3 the error of visible outlier configurations is only mitigated with bigger sets of training configurations. While these exceptions are interesting to inspect, our results show that our automated machine-learning pipeline for NFP prediction performs well in most cases.

RQ2: Model Training and Prediction Time. The training and prediction times are shown in Table 3. The average training time is 4.76 seconds, with a median of 2.21 seconds. The prediction is faster, with an average of 0.03 seconds and a median of 0.02 per predicted configuration. We argue that these times make NFP prediction using AutoML feasible since a) the training times are relatively low, and b) once a suitable ensemble has been trained, this can then be used for many, very fast predictions.

Additionally, while we did not do this for this evaluation, technically new information from previously not measured configurations could be used to improve the ensemble later on. This adaptation further improves the accuracy without having to train machine learning models again from the start, saving time in the process.

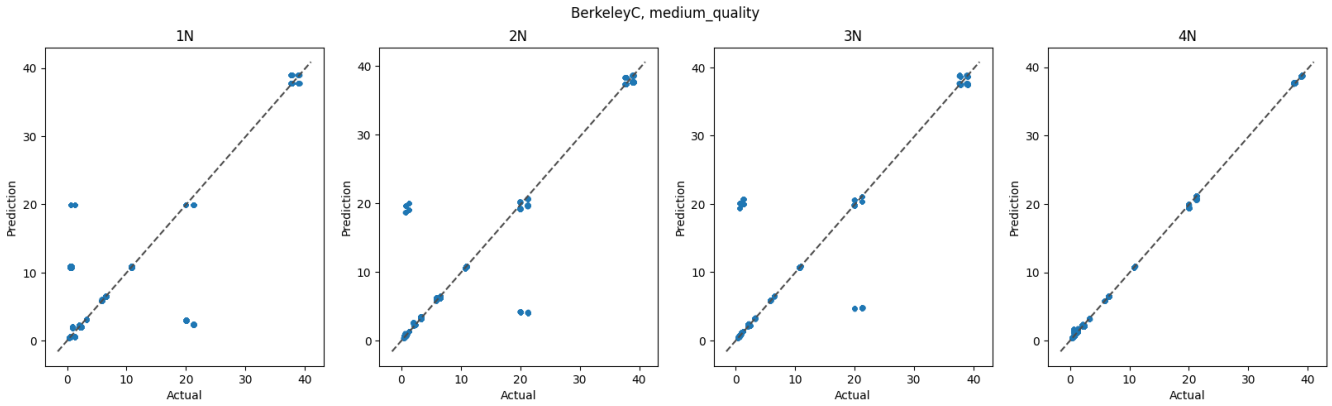

Figure 3: Prediction vs. actual performance for Berkeley (C) for different training set sizes (N to 4N)

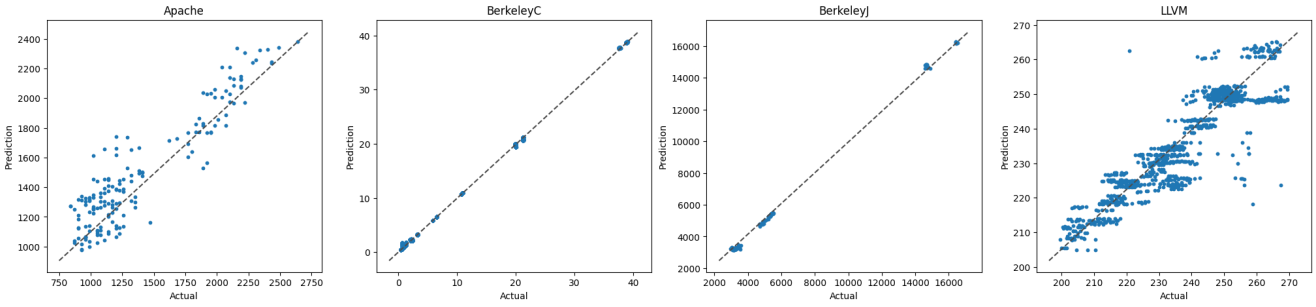

Figure 4: Prediction vs. actual performance comparison for the 4 subject systems with training set size 4N

System	_N	Training Time	Prediction Time	MAPE
Apache	1N	4.047	0.029	25.27
	2N	5.069	0.035	26.00
	3N	2.782	0.006	14.09
	4N	5.347	0.036	10.29
BerkeleyC	1N	25.545	0.01	295.88
	2N	2.173	0.038	121.64
	3N	1.895	0.067	121.28
	4N	2.76	0.041	1.48
BerkeleyJ	1N	2.25	0.009	5.32
	2N	2.096	0.065	3.61
	3N	1.989	0.047	2.00
	4N	23.944	0.045	1.48
LLVM	1N	1.141	0.007	6.25
	2N	1.33	0.002	4.29
	3N	4.01	0.037	3.825
	4N	3.138	0.011	2.19

Table 3: Training times, prediction times (in s), and MAPEs for the subject systems with different training data sizes

Threats to Validity

Internal Validity. We took *AutoGluon*'s `medium_quality` preset for training the machine learning models. It is the fastest preset, therefore producing low runtime measurements. However, the choice of the presets presents a tradeoff between run time and prediction quality. Using different presets or configuring *AutoGluon* further would influence the finding presented here. However, as our main use case was the ease of use compared to other NFP prediction approaches, we argue that using this preset represents automated machine learning well.

External Validity. Furthermore, we can not guarantee that the findings generalize to other subject systems. As the subject systems we chose are real systems from different domains, we assume the results to be generalizable to other applications.

5 CONCLUSION AND FUTURE WORK

In this work, we present a pipeline to use automated machine learning to predict non-functional properties for variants of configurable systems. The pipeline combines the prediction capabilities of machine learning and the automation of AutoML to be more accessible to end-users. The pipeline works by bringing configurations with their respective NFP value to the right input format, then training machine learning models with the configuration data, and outputting the trained model, which can then be used to predict NFP values for new configurations.

In an initial study, we show that the proposed pipeline is feasible to use, both regarding prediction accuracy and training and prediction time. However, not only the size of the training set influences the accuracy of the trained model, but also the subject system itself.

In future work, we aim to extend NFP prediction to variability models that do not exclusively include boolean configuration options, but also, for example, numerical attributes or feature multiplicities. This should be feasible because machine learning is agnostic of the underlying variability model. Additionally, for real-world applications, full configurations might not always be available and NFP indicators for partial configurations might already be helpful for decisions during the configuration process. Overall, we aim to use NFP prediction to improve sampling for complex variable systems. We aim to use NFP predictions to prioritize configurations that have to be tested, e.g., because of higher failure probability or vulnerability to attacks.

ACKNOWLEDGMENTS

We thank our reviewers for their constructive feedback. We also thank Darius Schefer for providing part of the implementation used in the feasibility study. This work has been partially funded by the German Research Foundation within the project *Co-InCyTe* (SCHA1635/15-1). We also thank the Ministry of Science, Research and Arts of the Federal State of Baden-Württemberg for the financial support of the projects within the Innovations Campus Future Mobility (ICM).

REFERENCES

[1] Manfred Broy. 2006. Challenges in automotive software engineering. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. Association for Computing Machinery, New York, NY, USA, 33–42. https://doi.org/10.1145/1134285.1134292

[2] Nick Erickson, Jonas W. Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alex Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *ArXiv* abs/2003.06505 (2020). https://api.semanticscholar.org/CorpusID:212725762

[3] Spandan Garg, Roshanak Zilouchian Moghaddam, Colin B. Clement, Neel Sundaresan, and Chen Wu. 2022. DeepDev-PERF: a deep learning-based approach for improving software performance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 948–958. https://doi.org/10.1145/3540250.3549096

[4] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. 2013. Variability-Aware Performance Prediction: A Statistical Learning Approach. 301–311. https://doi.org/10.1109/ASE.2013.6693089

[5] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wąsowski, and Huiqun Yu. 2018. Data-Efficient Performance Learning for Configurable Systems. 23, 3 (June 2018), 1826–1867. https://doi.org/10.1007/s10664-017-9573-6

[6] Lennart Holsten, Christian Frank, Jacob Krüger, and Thomas Leich. 2023. Electrics/Electronics Platforms in the Automotive Industry: Challenges and Directions for Variant-Rich Systems Engineering. In *Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems* (<conf-loc>, <city>Odense</city>, <country>Denmark</country>, </conf-loc>) *(VaMoS '23)*. Association for Computing Machinery, New York, NY, USA, 50–59. https://doi.org/10.1145/3571788.3571796

[7] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software* 182 (2021), 111044. https://doi.org/10.1016/j.jss.2021.111044

[8] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-Influence Models for Highly Configurable Systems. 284–294. https://doi.org/10.1145/2786805.2786845

[9] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting Performance via Automated Feature-Interaction Detection. 167–177.

[10] Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. 2013. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. 55, 3 (March 2013), 491–507. https://doi.org/10.1016/j.infsof.2012.07.020

[11] Lorenzo Vaccaro, Giuseppe Sansonetti, and Alessandro Micarelli. 2021. An Empirical Review of Automated Machine Learning. *Computers* 10, 1 (2021). https://doi.org/10.3390/computers10010011

[12] Len Wozniak and Paul Clements. 2015. How Automotive Engineering is Taking Product Line Engineering to the Extreme *(SPLC '15)*. 327–336. https://doi.org/10.1145/2791060.2791071