Master's Thesis

SAMPLING EFFECT OF NUMERIC OPTION ENCODING ON PERFORMANCE PREDICTION OF HIGHLY-CONFIGURABLE SOFTWARE SYSTEMS

ANNA-MARIA MAURER

October 25, 2022

Advisor:
Christian Kaltenecker Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering
Prof. Dr. Andreas Zeller CISPA Helmholtz Center for Information Security

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

ABSTRACT

Despite the fact that numeric features make up a huge share in real software systems, performance analyses often ignore their existence or process them in a binary-encoded format. However, there has been little research on the effect an encoding might have on performance predictions. Only one article has analyzed the effect of encoding on the learning aspect in more detail, but the sampling of configuration spaces has been disregarded completely.

This thesis fills that void by analyzing the effect of *one-hot encoding* versus numeric feature processing on the sampling phase, with the learning technique Random Forest. As effect metrics, we regard the duration and cycle time of the sampling and learning phases, in addition to the model prediction error. The findings are based on 22 case studies, of which 16 were measured from real software systems, and 6 were synthesized.

Our results demonstrate that there undoubtedly is a significant effect of encoding on the execution and outcome of the performance-prediction pipeline. For the sample size, processing numeric features leads to a smaller amount than applying binary encoding. The same implication applies to the duration of the sampling and learning phase. Due to the increase in sample size, the learning and total cycle time decreases when sampling over *one-hot encoded* features. In contrast to the duration, the prediction error is lower for binary than numeric feature encoding.

As a consequence, this thesis provides a first insight into the behavior of sampling for performance prediction when regarding numeric features in contrast to *one-hot encoding*.

ACKNOWLEDGMENTS

Writing a thesis is a very challenging task. Therefore, I am very thankful for the enormous support I received.

First, I would like to express my sincerest gratitude to my advisor Christian Kaltenecker, who guided me throughout this project and implemented required features in SPL Conqueror. Thank you for providing the topic to the thesis, as well as for your continuous aid and the occasional proofreading.

Furthermore, I would like to show my deep appreciation to Prof. Dr. Sven Apel for allowing me to write the thesis at his Chair, and always having time to discuss the direction of the thesis and the decisions made.

Additionally, I would like to recognize Prof. Dr. Andreas Zeller for performing the second examination of this thesis.

Finally, I could not have undertaken this journey without my family and friends. My thanks goes to Annabelle Weber for introducing me to the Chair and informing me about the open thesis topic. Special thanks also to my family for their patience and continuous encouragement.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ASE	Automated Software Engineering
BBD	Box-Behnken Design
CART	Classification and Regression Trees
CCID	Central Composite Inscribed Design
CPU	Central Processing Unit
DOD	D-Optimal Design
DFNN	Deep Feed-forward Neural Network
DT	Decision Tree
ESEC/FSE	European Software Engineering Conference and Symposium on the Foundations of Software Engineering
FF	Full-Factorial
FW	Feature-Wise
GAN	Generative Adversarial Network
GPR	Gaussian Process Regression
ICSE	International Conference on Software Engineering
KDE	Kernel Density Estimation
kNN	k-Nearest-Neighbors Regression
KRR	Kernel-Ridge Regression
LMT	Linear Model Tree
MAPE	Mean Absolute Percentage Error
MAE	Mean Absolute Error
ML	Machine Learning
MLR	Multiple Linear Regression
NN	Neural Network
NFP	Non-Functional Property
NOW	Negative Option-Wise
OFAT	One-Factor-at-a-Time
OW	Option-Wise
PBD	Plackett-Burman Design
PW	Pair-Wise
RF	Random Forest
RMSE	Root Mean Square Error

SE	Software Engineering
SHAP	SHapley Additive exPlanations
SVR	Support Vector Regression
tOW	*t*-Option-Wise

1 INTRODUCTION

Highly configurable software systems are deployed everywhere in nowadays life [39]. Thus, it is crucial to realize that the choice of *selected* features and feature values in a system highly influences its performance. This has far-reaching consequences, since the effect on the performance can occur in various ways and on many different aspects of a system [23]. Next to variations in the functionality, many non-functional properties (NFPs) are also affected [5]. It concerns e. g., runtime, Central Processing Unit (CPU) utilization, memory load, response time, or for zipping applications, compression rates.

Despite the fact that many software systems also use a fair share of numeric values to encode system options [8, 66], many analyses constrain themselves to assimilate only binary configuration options [29, 37, 43, 47, 57, 70, 88]. Those are often simpler to process, as binary features consist only of two states *selected* and *deselected*, whereas numeric options can encompass an arbitrary amount of numeric values, e. g., memory size.

To identify the optimal configuration regarding a certain NFP for a given software system, it is crucial to either know or estimate the performance of each valid configuration [86]. Since brute-force measuring is very inefficient for determining the performance of a software system [5], a vast majority of approaches specialize in performance predictions to locate the optimal configuration [57, 87].

Many articles deal with sampling and performance prediction. For those publications that include numeric options, there exist two strategies of integrating them inside the performance-prediction procedure [30]: On one hand, several articles accept numeric values when analyzing or predicting variable system performances. On the other hand, some publications rely on binary encoding methods to process numeric features. Hereby, each numeric option is transformed into one or multiple binary features, depending on the type of encoding. In this paper, we focus on the *one-hot encoding* technique, because it is widely accepted and considers each value of the numeric feature.

The applied encoding can have side effects on the performance analysis [30], as it plays a decisive factor when selecting sampling strategies [58], and binary encoding can lead to information loss. Nonetheless, nature and extent of this side effect have not been identified in detail yet, since there has been little research in this area prior to this thesis. To achieve an overview over the use of numeric and encoded features in performance prediction, we performed an extensive literature research. The procedure of binary encoding has been applied in quite some publications, with little reasoning and in some cases it was done implicitly. Other articles include numeric features directly, again seldom with a sound justification. Only three publications make use of both binary encoding and numeric processing, and contrast the results afterwards. However, the focus of two publications clearly lies in another area, and the choice of sampling strategies and the variation in learning are quite narrow [31, 34]. The third one generates insight into the effects encoding has on performance prediction. Thereby they concentrate on the learning aspect of predictions, and disregard the sampling completely [30].

Afterwards, we performed a quantitative analysis over 16 case studies based on real software systems, in addition to six synthetic case studies, to examine the effect of encoding in more detail. Thereby, we concentrated on the sampling phase of the prediction pipeline and analyze the sampling and learning duration, as well as the model prediction error. As a result, we gain insight of when the usage of *one-hot encoding* is preferable over numeric values.

In the evaluation, we showed the existence of a non-negligible effect on the performance prediction. We discovered that the duration of the steps and the whole pipeline, as well as the sample size, are significantly lower when applying numeric feature representation. In contrast, for the prediction error, *one-hot encoding* combined with learning of Random Forests performs better.

2 BACKGROUND

In this chapter, we explain the background knowledge needed to understand the mechanics used in this thesis. First, we introduce the basic terms of configurations and options. Later on, the foundations for the following chapters are lined out. Hereby, the focal point lies in elaborating the process for creating performance-prediction models, as well as the realization and variation of two crucial process steps. Additionally, it provides a short overview of the application used to perform these phases, and the statistical test included in the calculation of the results.

2.1 CONFIGURATION OPTIONS

Highly configurable systems offer a variety of configuration options to the user, which regulate what parts of a system are active and how the application is to proceed [5]. The choice for or against an option can influence both functionality and NFPs of a system [23, 92]. Those configuration options are also called features $\mathbf{f} \in \mathbf{F}$ and can take multiple forms.

The most common type of options are binary options $\mathbf{f}_{bin} \in \mathbf{F}_{bin}$, where the assigned value is an element of $\{0;1\}$ [5]. Hereby 1 signifies that the feature is *selected*, and 0 means that the option is *deselected*. In real-life systems, for instance they can be used to portray whether a log is to be written, the data should be compressed, or encryption shall be used. For the following explanations in the chapter, we use the 7 binary features:

$$A, B, C, D, E, F, G \in \{0;1\}$$

Another, more complex kind of configuration options are numeric options [58]. In contrast to binary ones, those allow a discrete amount of real numbers. A numeric feature $\mathbf{f}_{num} \in \mathbf{F}_{num}$ can for instance store buffer or block sizes, auto-save intervals or work memory capacity. Some examples, which will also be referred to later on, are:

$$M \in \{2;4;8;16;32\},$$

$$N \in \{-3;0;1;8;15\},$$

$$O \in \{1;3;5;7;9;11;17\}.$$

2.1.1 *Feature Model*

Moreover, configuration options are usually organized in a hierarchical structure, also called feature model [5]. They are written as Boolean formulas describing the interdependence of features. In there, we can portray child-parent relationships between options. Thus, if a parent feature is *selected*, a Boolean formula regulates the presence of child options. In contrast, for all *deselected* features, their sub-options are per default *deselected*.

Commonly, there exist different types of relationships, based on Boolean `AND`, `OR` and `XOR` operators [93]. For `OR` relations (and `XOR` respectively), at least (exactly) one of the children has to be *selected*. With the `AND` operator, all sub-features are in principle mandatory. However, in some scenarios optional sub-trees are of necessity. Therefore each direct child of an `AND` relation is supplemented with a unary operator signifying whether the option is obligatory or elective.

An example for a feature model is presented via Boolean Formula in Equation 2.1.

$$\begin{aligned} A \wedge (B \Rightarrow A) \wedge ((C \vee D) \Leftrightarrow B) \wedge (E \Leftrightarrow A) \wedge ((F \vee G) \Leftrightarrow E) \wedge (\neg(F \wedge G)) \\ \wedge (M = 2 \vee M = 4 \vee M = 8 \vee M = 16 \vee M = 32) \\ \wedge (N = -3 \vee N = 0 \vee N = 1 \vee N = 8 \vee N = 15) \\ \wedge (O = 1 \vee O = 3 \vee O = 5 \vee O = 7 \vee O = 9 \vee O = 11 \vee O = 17) \end{aligned} \tag{2.1}$$

It portrays the following relationships of a feature model:

- If option A is *selected* (and as the root node, it always is), E is mandatory and B can or cannot be *selected*. Based on A, M, N and O can take any of their defined values.
- If B is *deselected*, then neither C or D can be *selected*. Otherwise, at least one of them must be *selected*.
- Similarly, the presence of E dictates that exactly one of F and G must be *selected*.

As an alternative to Boolean formulas, feature models are usually displayed in form of a tree-based graph, so that they are easier for humans to read [5]. Those graphs are then called feature diagrams.

Figure 2.1 displays an exemplary feature diagram equivalent to the feature model in Equation 2.1.

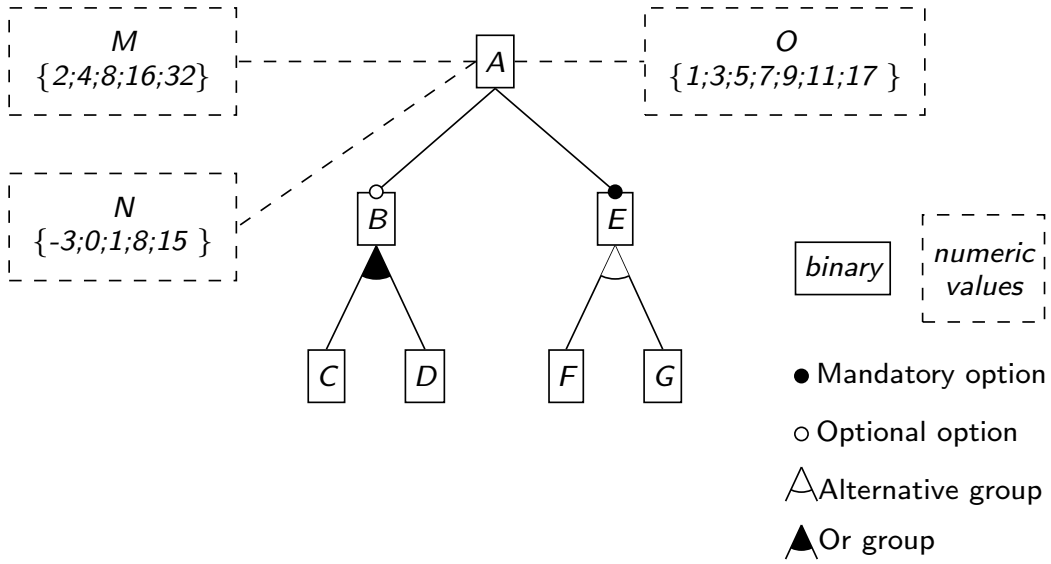


Figure 2.1: Example of a feature diagram with 7 binary options A to G and three numeric options M, N, O

2.1.2 *Configurations*

A configuration $\mathbf{c}$ is then made up of features $\mathbf{f}$ and their assigned values $\mathbf{c}(\mathbf{f})$ [37]. For instance, regard the following configuration given as a set of feature-value pairs:

$$\mathbf{c}_{example} = \{A = 1; B = 1; C = 1; D = 1; E = 1; F = 0; G = 1; M = 2; N = 1; O = 11\}.$$

The set of all possible configurations $\mathbf{C}$ is calculated via the Cartesian product of all option ranges [42]. In our case, the amount of entries equals

$$|\mathbf{C}| = \mathcal{P}(\mathbf{F}) = 2^{|\mathbf{F}_{bin}|} \cdot |M| \cdot |N| \cdot |O| = 2^7 \cdot 5 \cdot 5 \cdot 7 = 22\,400.$$

Even though, depending on the construction of the feature model, the majority of those configurations may be invalid [85]. For our example, the usable configurations in $\mathbf{C}_{fm}$ amount to 6,25%:

$$\begin{aligned} |\mathbf{C}_{fm}| &= 1_{A=1} \cdot (1_{B=0} + 1_{B=1} \cdot (1_{C=1;D=0} + 1_{C=0;D=1} + 1_{C=1;D=1})) \\ &\quad \cdot 1_{E=1} \cdot (1_{F=1;G=0} + 1_{F=0;G=1}) \cdot |M| \cdot |N| \cdot |O| \\ &= 1 \cdot 4 \cdot 1 \cdot 2 \cdot 5 \cdot 5 \cdot 7 = 1\,400 = 6,25\% \cdot |\mathbf{C}|. \end{aligned}$$

In addition to the restrictions given by the options' hierarchy, Boolean constraints appended to the feature model can limit the set of accepted configurations even further [62]. They are commonly called cross-tree constraints [5]. For instance, regard an additional constraint $con := G \Rightarrow (M > 8)$ as to be included into our feature model. Then, the number of valid configurations $\mathbf{C}_{fm \wedge con}$ is limited to 4.375%:

$$\begin{aligned} |\mathbf{C}_{fm \wedge con}| &= 1_{A=1} \cdot (1_{B=0} + 1_{B=1} \cdot (1_{C=1;D=0} + 1_{C=0;D=1} + 1_{C=1;D=1})) \\ &\quad \cdot 1_{E=1} \cdot (1_{F=1;G=0} \cdot |M| + 1_{F=0;G=1} \cdot |(M > 8)|) \cdot |N| \cdot |O| \\ &= 1 \cdot 4 \cdot 1 \cdot (5 + 2) \cdot 5 \cdot 7 = 1\,120 = 4.375\% \cdot |\mathbf{C}|. \end{aligned}$$

The configuration space is defined as the set of all valid configurations given by the whole feature model (in our case $\mathbf{C}_{fm \wedge con}$).

2.1.3 *Binary Encoding for Numeric Features*

As not every approach is capable of processing numeric input, some approaches transform the numeric features into one or multiple binary options by specified encoding mechanisms. There exist several strategies for such a binary encoding, depending on the granularity of the wanted result.

The most frequently used technique is commonly called *one-hot encoding*. The first introduction of this encoding on feature models occurs in Grebhahn et al. by the name *discretization* [31, 34]. *One-hot encoding* transforms each numeric option into a tree consisting of binary features with height 1. Hereby the mandatory tree root node takes the place of the numeric option. Each numeric value is represented by an annotated binary child node in the tree. Its selection corresponds to the feature taking this concrete value. A numeric option can only take one value in a configuration, therefore, the corresponding binary

nodes are alternative (XOR). Figure 2.2 visualizes the scheme of this encoding. We call the set of all *one-hot encoded* options $\mathbf{F}_{ohe}$, this set contains the whole sub tree created by the encoding including their root features. The *one-hot encoding* transformation is bijective, as every numeric value can be depicted by a unique selection of binary features; and each selection of the *one-hot encoded* features can be re-transferred to a numeric value.

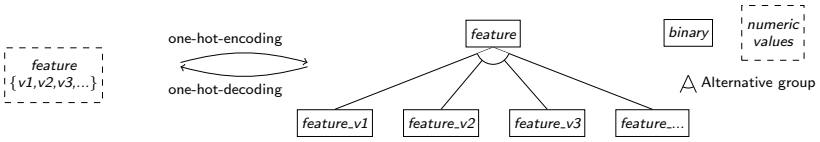


Figure 2.2: Scheme describing the process of *one-hot encoding* and its reversal, *one-hot decoding*

The Boolean feature model in Equation 2.2 shows the model introduced in Section 2.1.1, where the numeric option M has been *one-hot encoded*. In this case $\mathbf{F}_{ohe}$ is given as:

$$\mathbf{F}_{ohe} = \{M; M\_2; M\_4; M\_8; M\_16; M\_32\}$$

Although usually all numeric features are encoded in the same way (all *one-hot encoded* or all numeric), for simplification reasons we transform only M in this example. All changes were rendered bold for an easier identification. Furthermore, the additional constraint introduced in the previous section is left out.

$$\begin{gathered}
A \wedge (B \Rightarrow A) \wedge ((C \vee D) \Leftrightarrow B) \wedge (E \Leftrightarrow A) \wedge ((F \vee G) \Leftrightarrow E) \wedge (\neg(F \wedge G)) \\
\wedge(N = -3 \vee N = 0 \vee N = 1 \vee N = 8 \vee N = 15) \\
\wedge(O = 1 \vee O = 3 \vee O = 5 \vee O = 7 \vee O = 9 \vee O = 11 \vee O = 17) \\
\wedge(\mathbf{M} \Leftrightarrow \mathbf{A}) \wedge ((\mathbf{M_2} \vee \mathbf{M_4} \vee \mathbf{M_8} \vee \mathbf{M_16} \vee \mathbf{M_32}) \Leftrightarrow \mathbf{M}) \\
\wedge(\neg(\mathbf{M_2} \wedge \mathbf{M_4})) \wedge (\neg(\mathbf{M_2} \wedge \mathbf{M_8})) \wedge (\neg(\mathbf{M_2} \wedge \mathbf{M_16})) \wedge (\neg(\mathbf{M_2} \wedge \mathbf{M_32})) \\
\wedge(\neg(\mathbf{M_4} \wedge \mathbf{M_8})) \wedge (\neg(\mathbf{M_4} \wedge \mathbf{M_16})) \wedge (\neg(\mathbf{M_4} \wedge \mathbf{M_32})) \\
\wedge(\neg(\mathbf{M_8} \wedge \mathbf{M_16})) \wedge (\neg(\mathbf{M_8} \wedge \mathbf{M_32})) \wedge (\neg(\mathbf{M_16} \wedge \mathbf{M_32}))
\end{gathered} \tag{2.2}$$

The preceding feature model corresponds to the feature diagram presented in Figure 2.3.

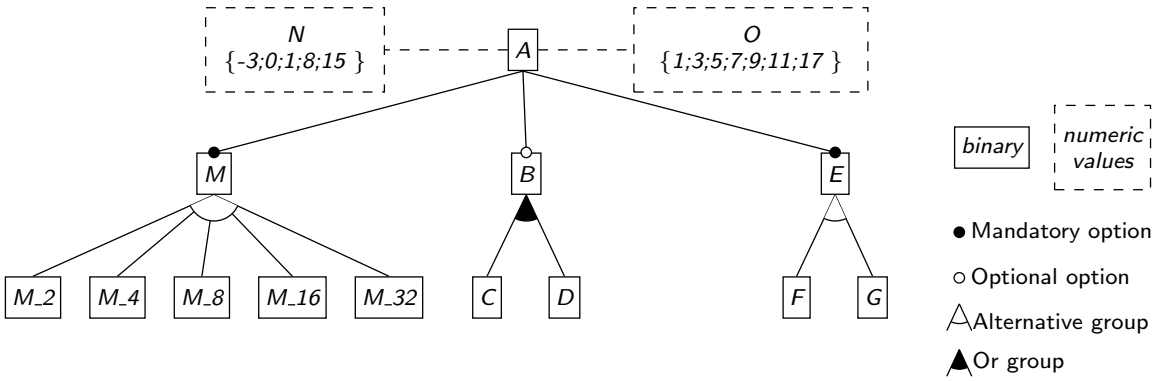


Figure 2.3: Feature diagram from Figure 2.1, where option M has been *one-hot encoded*

Furthermore, other encoding techniques can be realized by disregarding some feature values and thus, possibly valid configurations. Denk, for instance proposes to ignore all

but the default value [20]. For the categorical variable *SQLITE_THREADSAFE* encoded by numbers all but the two most contrasting options where left out. Similarly, Jamshidi et al. and Velez et al. suggest the use of just two values, typically opposites on the feature value spectrum [43, 88].

As those techniques restrict the configuration space and are seldom applied in research publications, we further on concentrate on *one-hot encoding* and its effects.

2.1.4 *Feature Interactions*

Options do not only exist in isolation, but rather they also must interact with each other to reach their respective goals [5]. This also affects the NFPs, as the actual measurement $\mathbf{NFP}(\mathbf{c})$ for a configuration $\mathbf{c}$ usually differs from the value resulting from adding the influence $\mathbf{i}(\mathbf{f} = \mathbf{c}(\mathbf{f}))$ of each single feature $\mathbf{f}$ to the base term $\mathbf{NFP}_0$.

$$\mathbf{NFP}(\mathbf{c}) \neq \mathbf{NFP}_0 + \textstyle\sum_{\mathbf{f} \in \mathbf{F}} \mathbf{i}(\mathbf{f} = \mathbf{c}(\mathbf{f}))$$

One popular example is a mailing system which allows users to both encrypt and compress mails [6, 60]. Regarding the duration, each of those features individually adds an effect greater than 0. However, combined the supplement is much smaller than both option influences together, as the compression reduces the amount of encryption that needs to be performed by the system. This shows that the interaction between compression and encryption also has an impact on the duration.

$$\begin{aligned}\mathbf{NFP}(\mathbf{c}) &= \mathbf{NFP}_0 + \mathbf{i}(\text{🔒} = 1) + \mathbf{i}(\text{🗜} = 1) + \mathbf{i}(\text{🗜} = 1, \text{🔒} = 1) \\ &< \mathbf{NFP}_0 + \mathbf{i}(\text{🔒} = 1) + \mathbf{i}(\text{🗜} = 1)\end{aligned}$$

Similarly, the interactions of various features may affect several NFPs in other scenarios [6]. Those can be of any order o, therefore all combinations of o options may be relevant as well having either a positive or negative effect on the NFP. These effects should be covered when building models. One example for a high-order interaction is the performance feature bug portrayed in Nguyen et al. concerning 6 options [67].

2.2 PERFORMANCE-PREDICTION PIPELINE

The regarded pipeline for performance predictions is an adaption of the pipeline presented by Kaltenecker et al. and consists of four central steps [46]. This analysis has no insight into the implementation of a configurable system, therefore it represents a black-box approach [89].

(1) *Sampling configurations* selects a subset of all valid configurations based predefined strategies [35]. This step is essential, as the amount of configurations to consider increases exponentially when adding new features. Consequently, performing the later actions for all valid configurations would be quite inefficient [63]. The choice of sampling strategies to apply is mostly based on the option encoding, thus the feature setup may affect the final sample. The resulting sample set of this phase forms the foundation of all following steps.

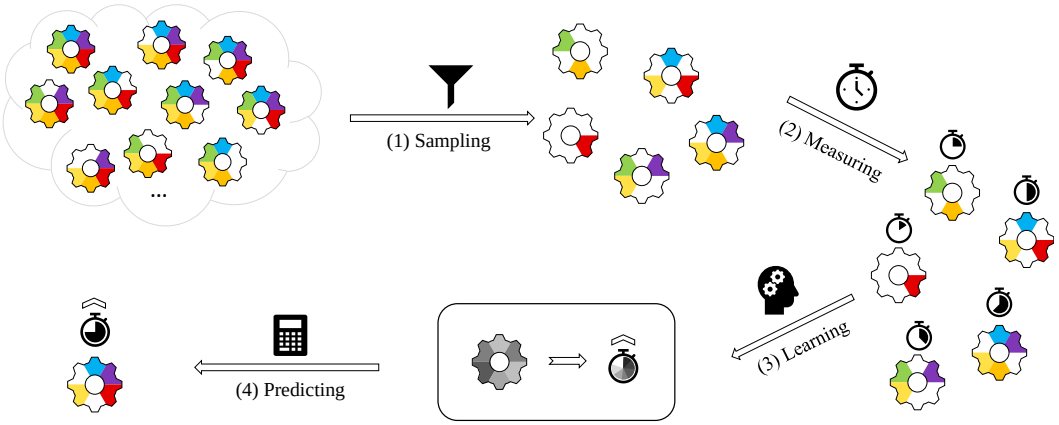


Figure 2.4: Pipeline used for performance prediction
It includes its four steps (1) sampling, (2) measuring, (3) learning, and (4) predicting.

(2) *Measuring performance* determines the property values of the sampled configurations for the examined software system on a constant setting [33]. This step was left out in the creation of this thesis, as the measurements for the whole configuration set were provided by the Chair. In practice, this phase takes up the majority of the prediction duration when estimating NFP for a new software system [63].

(3) *Learning models* creates a property model with machine-learning techniques based on the sample set and their corresponding measurements, taking into account all features and their interactions [89]. The inherent assumptions may cause enormous variation in the model quality, thus the modeling parameters, also referred to as hyperparameters, have to be chosen well [33, 38]. The learning process is executed upon sampled data, therefore the size and encoding of the sample set affects the quality of the model as well [26].

(4) *Predicting*: With the resulting model, the regarded NFP can be estimated for all other configurations. The further goal is usually the optimization of the performance model to identify the (nearly) optimal configuration [63, 68].

The central steps for this thesis, sampling and learning, are elaborated in Section 2.4 respectively Section 2.5, and their execution is described in Section 2.3.

ENCODING-INDEPENDENCE Regarding the sampling and learning steps of the prediction pipeline, it is crucial to realize that the encoding used for sampling e_s does not necessarily have to be identical to the one applied for learning e_l. This presents a greater opportunity for optimization, as on one hand we can choose an encoding together with a suitable strategy for sampling, on the other hand for learning we can select the encoding and an appropriate machine-learning technique. However, similar to function composition, there exist prerequisites for the transformations between the encoding strategies $t_{\_,s}$ and $t_{s,l}$:

- The codomain of $t_{\_,s}$ and the domain of the $t_{s,l}$ function must match, so that every value resulting from $t_{\_,s}$ is applicable to $t_{s,l}$ [83].

- If a re-transfer to the original values is intended, bijection is recommended for both $t_{\_,s}$ and $t_{s,l}$ [19].

One-hot encoded and numeric representation are two encoding strategies, which along with their transformations *one-hot encoding* and *one-hot decoding* fit all these criteria. Therefore, *one-hot encoded* and numeric representation are allowed for both the sampling and learning phase in our experiment.

2.3 SPL CONQUEROR

Two of the phases inside the performance-prediction pipeline, namely sampling and learning, can be realized partially with the application *SPL Conqueror*[1]. It can be used to sample configurations including measurements and learn performance models to make predictions.

In SPL Conqueror software systems are regarded as black-box models, as deeper analysis like in Velez et al. [87] requires available and processed source code, which is only given for open-source projects.

After specifying the variability model and the valid configurations including performance measurements, the sampling strategies are to be selected. The following Table 2.1 displays the strategies offered by SPL Conqueror that we consider to use.

Table 2.1: Sampling strategies supported by SPL Conqueror for both binary and numeric features, as far as they are relevant in the progress of this thesis.
Equivalents between sampling strategies are marked by an equivalence symbol (⇔).
The entries in brackets are also mechanisms provided by SPL Conqueror, even though they are not actually sampling techniques as they just return all configurations.

Binary Sampling Strategies		Numeric Sampling Strategies
Option-Wise (OW)	⇔	One-Factor-at-a-Time (OFAT)
Negative Option-Wise (NOW)		Plackett-Burman Design (PBD)
Pair-Wise (PW)		Box-Behnken Design (BBD)
t-Option-Wise (tOW)		Central Composite Inscribed Design (CCID)
Random	⇔	Random
Distance-based		
(Allbinary)	⇔	(Full-Factorial (FF))

In the command line version, the options over which to sample can be indicated as well, so that the sampling occurs only on a subset of all potentially relevant options. This program aspect will be used to split the binary from the *one-hot encoded* features later on (cf. Section 4.3.1). The results of the sampling step can be issued for separate analysis.

Afterwards, the model learning can also be executed in SPL Conqueror. The NFP to estimate can be selected, and the learning algorithm including additional parameters may be specified for the sampled set. SPL Conqueror natively supports only Multiple Linear Regression (MLR), but other techniques are available via python imports of the package

1 available at: https://github.com/se-sic/SPLConqueror/tree/xgboost, last visited at: 2022-08-12.

scikit-learn[2] to perform the training and predictions needed. Further commands allow the optimization of the model and more detailed analysis of the learning process.

2.4 SAMPLING STRATEGIES

The available sampling strategies highly depend on the given feature type. Most established sampling strategies can only be used with binary features, whereas many experimental designs are applicable for sampling numeric features [79].

Only few strategies are capable of processing both binary and numeric options. Just one is considered in this thesis, random sampling [38, 61, 78]. Despite that possibility, SPL Conqueror applies random sampling separately per datatype and joins the results [46]. Therefore the strategy is described for both binary and numeric input individually.

2.4.1 *Binary Sampling Methods*

There are various sampling strategies in use for binary features. For illustration purposes, the options A to G including feature model constraints are consulted in this section, as introduced in Section 2.1. All valid configurations $\mathbf{C}_{bin}$ regarding those features are portrayed in Table 2.2.

Table 2.2: All valid configurations $\mathbf{C}_{bin}$ based on the binary features A to G

$\mathbf{c}_{index}$	A	B	C	D	E	F	G
1	1	0	0	0	1	1	0
2	1	1	0	1	1	0	1
3	1	1	0	1	1	1	0
4	1	1	1	0	1	0	1
5	1	1	1	0	1	1	0
6	1	1	1	1	1	0	1
7	1	1	1	1	1	1	0

One category of sampling strategies are coverage-based strategies, as they are defined by a coverage criteria [90]. In this thesis, the granularity of coverage is restricted to the feature-level, since SPL Conqueror analyzes black-box models.

One member of this category is Option-Wise (OW) sampling, also called Feature-Wise (FW) sampling, where for each available feature a valid configuration with this feature *selected* and minimal other options *selected* is chosen [35]. This minimizes the effect of interactions in later analyses. For our example, this would result for instance in the set $\mathbf{C}_{OW} = \{\mathbf{c}_1; \mathbf{c}_2; \mathbf{c}_3; \mathbf{c}_4; \mathbf{c}_5\}$, reasoning and iteration cycles are presented in Table 2.3.

In contrast, Negative Option-Wise (NOW) maximizes the amount of interactions to consider, as here valid configurations are selected with a maximum of *selected* features while

2 https://scikit-learn.org/, last visited at: 2022-07-23.

deselected the current option for each feature of interest [32]. An example is $\mathbf{C}_{NOW} = \{\mathbf{c}_1; \mathbf{c}_2; \mathbf{c}_5; \mathbf{c}_6; \mathbf{c}_7\}$, as clarified by Table 2.4.

Table 2.3: An exemplary result for OW sampling based on the binary features A to G

$\mathbf{c}_{index}$	A	B	C	D	E	F	G
1	1	0	0	0	1	1	0
5	1	1	1	0	1	1	0
4	1	1	1	0	1	0	1
3	1	1	0	1	1	1	0
1	1	0	0	0	1	1	0
1	1	0	0	0	1	1	0
2	1	1	0	1	1	0	1

Table 2.4: In contrast Negative Option-Wise (NOW) sampling for the same features

$\mathbf{c}_{index}$	A	B	C	D	E	F	G
-	0	-	-	-	-	-	-
1	1	0	0	0	1	1	0
2	1	1	0	1	1	0	1
5	1	1	1	0	1	1	0
-	-	-	-	-	0	-	-
6	1	1	1	1	1	0	1
7	1	1	1	1	1	1	0

As the result is usually iteratively determined, the focal feature per row is underlined.

Table 2.5: tOW sampling for $t = 2$ (PW) and $t = 3$
In both tables, the two respective three underlined features per line indicate the combinations covered by this configuration. The mandatory options A and E are only considered in the first row, as including them in other iteration is equivalent to performing OW and PW on the other features.

(a) PW Sampling

$\mathbf{c}_{index}$	A	B	C	D	E	F	G
1	1	0	0	0	1	1	0
4	1	1	1	0	1	0	1
3	1	1	0	1	1	1	0
5	1	1	1	0	1	1	0
2	1	1	0	1	1	0	1
6	1	1	1	1	1	0	1
5	1	1	1	0	1	1	0
4	1	1	1	0	1	0	1
3	1	1	0	1	1	1	0
2	1	1	0	1	1	0	1

(b) tOW Sampling for $t = 3$

$\mathbf{c}_{index}$	A	B	C	D	E	F	G
1	1	0	0	0	1	1	0
6	1	1	1	1	1	0	1
5	1	1	1	0	1	1	0
4	1	1	1	0	1	0	1
3	1	1	0	1	1	1	0
2	1	1	0	1	1	0	1
7	1	1	1	1	1	1	0
6	1	1	1	1	1	0	1

Furthermore, t-Option-Wise (tOW) has been designed to recognize interactions of size t or less, thus all combinations of t *selected* options are required [46]. However, this strategy becomes quite costly with increasing t, as the size of the resulting set lies in $O(|\mathbf{F}_{bin}|^t)$ [46]. The special case of $t = 2$ is also called Pair-Wise (PW) [31]. Table 2.5 shows exemplary iterations

for PW and tOW with $t = 3$, which result in $\mathbf{C}_{PW} = \{\mathbf{c}_1; \mathbf{c}_2; \mathbf{c}_3; \mathbf{c}_4; \mathbf{c}_5; \mathbf{c}_6\}$ and $\mathbf{C}_{3OW} = \mathbf{C}_{bin}$ respectively. The small number for both is a consequence of the tight constraints, in a setting with 7 optional and independent features, the number of sampled configurations would amount to 21 respective 35.

A more randomized approach is given by distance-based sampling [47]. Here, an origin (all options *deselected*), and a distance metric (Manhattan Distance) for all configurations is defined. Based on this setting, a random distance d is chosen. Afterwards, a configuration with distance d to the origin is randomly selected by a SAT-Solver. An advantage of this strategy is the flexible amount of samples selected. Moreover, the resulting configurations are more (randomly) widespread than pure SAT-based approaches, as with those the sample set often forms dense clusters inside the total configuration space [69]. Table 2.7 provides an example with 4 configurations.

Table 2.7: Example of 4 configurations selected via distance-based sampling.
Since the origin implies that all features are *deselected*, a distance with valid configurations lies within $\{3; 5; 6\}$.
All non-mandatory features are underlined when *selected* for this configuration.

origin:

A	B	C	D	E	F	G
0	0	0	0	0	0	0

$\mathbf{c}_{index}$	Distance	A	B	C	D	E	F	G
1	3	1	0	0	0	1	1	0
4	5	1	1	1	0	1	0	1
2	5	1	1	0	1	1	0	1
6	6	1	1	1	1	1	0	1

The idea of random sampling is, as the name indicates, a pure random selection of a predefined number of samples among the valid configuration space. However, the systems often have too many possible configurations to enumerate, and many of those may be invalid [53]. This is visualized by an example of the Linux kernel: Out of one million randomly selected configurations, less than 100 were valid. Thus, it is very inefficient to randomly generate a configuration and afterwards check it for validity. Therefore, in practice SAT solvers identify all valid configurations [79]. Based on this result set, an arbitrary amount of valid configurations can be chosen.

The sampling *allbinary* choice in SPL Conqueror (cf. Table 2.1) defines the whole configuration set as sampled. Undoubtedly, this strategy is very inefficient and thus seldom applied in practice [36]. In our case, as the measurements have already been completed, using all data might provide some insights for interpreting other sampling strategies. Nonetheless, the results of those runs do not take part in any evaluation.

2.4.2 *Numeric Sampling Methods*

In this section, our example features from Section 2.1 are accessed once again; this time the numeric ones M, N, O. As no constraints are defined between those, all $|\mathbf{C}_{num}| = 5 \cdot 5 \cdot 7 = 175$ configurations are considered valid for this subsection.

Three strategies available have equivalents in binary sampling: Just like OW, One-Factor-at-a-Time (OFAT) varies one feature while leaving the others in default value (usually minimum or 0) [33]. Visually interpreted, the selected configurations try to cover all points on the option axes. Figure 2.5 illustrates this aspect, and Table 2.8 lists all corresponding configurations for the given options.

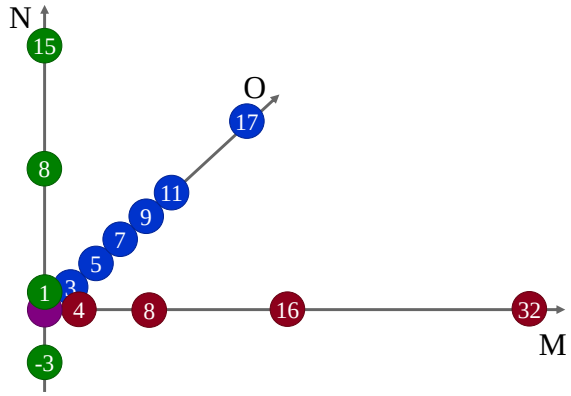


Figure 2.5: Sample for M, N, O with OFAT
The number in the blue dots marks the value for the feature on the axis it lies, the other values are default.

Table 2.8: Configurations for the example features M, N, O when using OFAT

M	N	O	M	N	O
4	0	1	2	0	3
8	0	1	2	0	5
16	0	1	2	0	7
32	0	1	2	0	9
2	-3	1	2	0	11
2	1	1	2	0	17
2	8	1	center:		
2	15	1	2	0	1

Full-Factorial (FF) selects the combination of every value from each option, thus all possible configurations are chosen [45]. It corresponds to the *allbinary* choice in binary sampling, and the same rules are applied here. The random sampling for numeric options is identical to the one of binary features [79].

Various experimental designs have been transferred as sampling strategies for numeric features. These techniques frequently make use of attributions to match the values of k different features onto l homogeneous levels, based on which the design then is sampling over the given scenario. Afterwards, the mapped samples are transformed into real configurations referring now again to feature values.

One adopted design is the Plackett-Burman Design (PBD) introduced by Plackett and Burman [73]. Its advantage is the flexible number of levels l and configurations to extract, which are also called measurements m [35]. Hereby, m must be no higher than $l^{|\mathbf{F}_{num}|}$, as the number of levels constricts the configuration space considered by the sampling strategy. Nonetheless, due to conventions, m often takes the form of $l^x, 0 < x \leq |\mathbf{F}_{num}|$.

First, the levels $0, 1, \ldots, l-1$ are mapped uniformly to each option range, such that 0 points to the smallest value, and $l-1$ to the largest. Based on this allocation, a random array of size $m-1$ is created for the first feature [73].

For all later features, this array is shifted by one value compared to the previous array. Finally, the default configuration (minimum of all option values) is added as the last row.

An example showing the mapping including array shift with $l = 3, m = 9$ is displayed in Figure 2.6. Moreover, in Table 2.9 we illustrate a realistic sample from the numeric features introduced in Section 2.1.

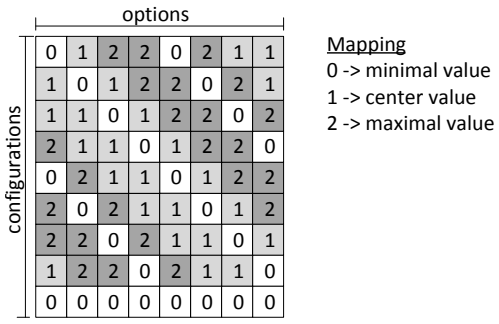


Figure 2.6: Exemplary sample of mappings with PBD for $l = 3$ and $m = 9$ measurements [35]

Table 2.9: Configurations for the example features M, N, O for $l = 3$ and $m = 9$ sampled with PBD as the tree first columns of Figure 2.6

M	N	O	M	N	O
2	1	17	32	-3	17
8	-3	7	32	15	1
8	1	1	8	15	17
32	1	7	2	-3	1
2	15	7			

Box-Behnken Design (BBD) is a second-order design published by Box and Behnken [11]. It is based on three option levels (-1 for the minimal value, 0 refers to the mid-value and 1 maps to the highest value) [33]. Depending on an internal variable k, a fractional design consisting of multiple blocks is created. In each block, one part of the features is set to 0, whereas the remaining m options form a full-factorial mesh of the levels -1 and 1. In addition, some center points are selected. Figure 2.7 provides the standard design for three features, and additionally Table 2.10 contains sampled configurations for our example.

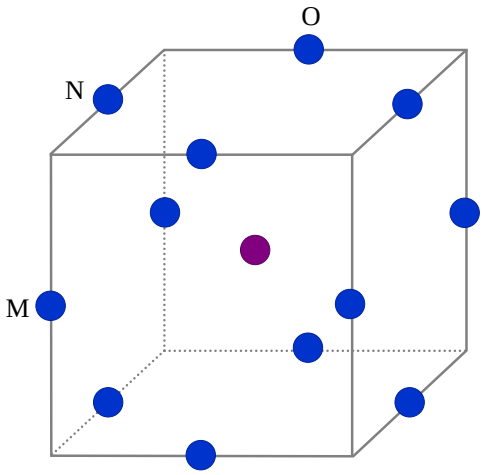
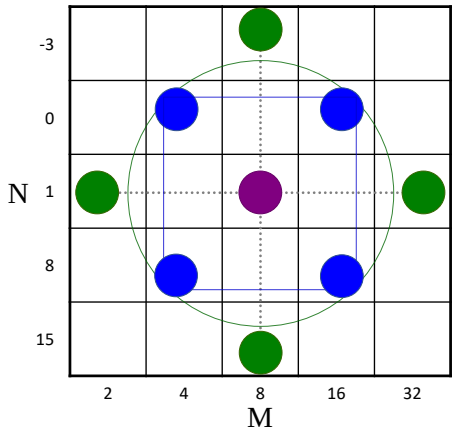


Figure 2.7: Design scheme for BBD with 3 features It can be interpreted as a cube, from which all midpoints of the edges are selected, together with the cube's center [2].

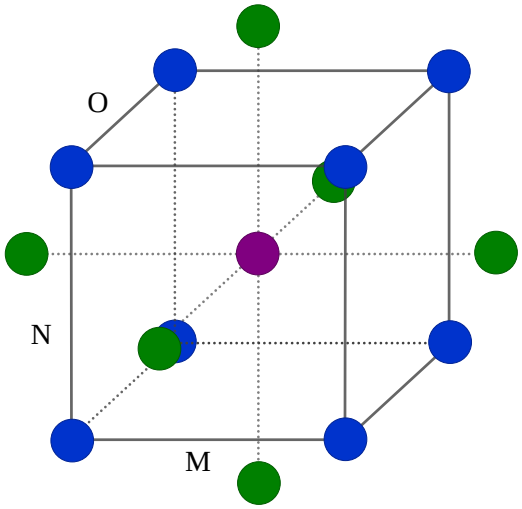
Table 2.10: An exemplary sampling with BBD for the three options (M, N and O) based on the design scheme portrayed in Figure 2.7.

M	N	O	M	N	O
32	1	17	2	1	17
32	1	1	2	1	1
32	15	7	2	15	7
32	-3	7	2	-3	7
8	15	17	8	-3	17
8	15	1	8	-3	1
	center:		8	1	7

Central composite design is a second-order technique presented in 1951 by Box and Wilson [12]. It is realized by embedded factorial designs, combined with $2 \cdot k$ star points and additional center points. Those are projected into a mapped feature space, with their origin being in the middle of the value range [32].



(a) Design Scheme with two Options M and N
The star (in green) is shaped like a cross and the blue embedded factorial design looks like a square [45].



(b) Three-dimensional Scheme with M, N and O
The factorial is represented by a small die of blue dots. It is surrounded by 6 green points located on each end of the option axes. In contrast to BBD, these points don't mark midpoint edges, instead they are situated in the center of each side of the feature cube [12].

Figure 2.8: CCID design schemes for two and three options

Table 2.11: Central Composite Design in two examples for 2 and 3 features respectively
The green star dots are listed on the left side of each table, the blue factorial on the right side.

M	N	M	N
2	1	4	0
8	-3	4	8
8	15	16	0
32	1	16	8
center:		8	1

(a) Sampling over 2 options M and N

M	N	O	M	N	O
2	1	7	4	0	3
8	-3	7	4	0	11
8	1	1	4	8	3
8	1	17	4	8	11
8	15	7	16	0	3
32	1	7	16	0	11
center:			16	8	3
8	1	7	16	8	11

(b) Listing of samples for three features M, N and O

There exist multiple variations of this design. In this thesis, we use Central Composite Inscribed Design (CCID), a central design using five option levels. The mapping is similar to BBD. In contrast to the central composite circumscribed design, the other two levels regarded in CCID are found inside the feature range [94]. The end points of the star lie at the outer ring of the feature space; for each option the minimum and maximum are combined with the center value of the other features. Concerning the factorial design, its corners have about the same distance to the center as the star points, and they are stationed diagonally from the feature axes. Figure 2.8 shows two design schemes for two respective three features, and Table 2.11 displays the configurations on our example with once just M and N as well as another with all three options.

2.4.3 *Combination of Sample Sets*

When sampling over different feature subsets, the resulting partial configurations have to be merged to receive usable configurations over the whole feature space [32]. To prevent information losses during the join, it is realized via a Cartesian product [35], which combines each sampled configuration of the first set with every entry from the second sampling result. For m partial configurations sampled over the first subset of features, and respectively n for the second feature subset, the product issues $m \cdot n$ complete configurations. For our example, when using the results from distance-based sampling $\mathbf{C}_{DB}$ presented in Table 2.7 merged with the samples generated by PBD $\mathbf{C}_{PBD}$ displayed in Table 2.9, the possible configurations amount to 36 as calculated in Equation 2.3.

$$|\mathbf{C}_{DB\times PBD}| = |\mathbf{C}_{DB}| \cdot |\mathbf{C}_{PBD}| = 4 \cdot 9 = 36 \tag{2.3}$$

However, a feature model usually incorporates restrictions to the configuration space [8] – given by the hierarchical setup of the options, as well as additionally formulated constraints (Section 2.1.1 and Section 2.1.2). Consequently, not all of the resulting configurations are necessarily valid, and each configuration emitted by the Cartesian product has to be examined for validity. Therefore, the amount of configurations to consider may be significantly lower than the configurations identified by the Cartesian product, especially if the hierarchy is complex or the additional constraints limit the configuration space drastically. In our example of Section 2.1.2, the feature M is restricted by the *selection* of G. This excludes 18 entries as calculated in Equation 2.4 and leaves the 18 valid configurations $\mathbf{C}_{DB\times PBD;valid}$ as stated in Table 2.13.

$$\begin{aligned} |\mathbf{C}_{DB\times PBD;valid}| &= |\mathbf{C}_{DB\times PBD}| - |\{\mathbf{c} \mid \mathbf{c} \in \mathbf{C}_{DB} \wedge \mathbf{c}(G) = 1\}| \\ &\quad \cdot |\{\mathbf{c} \mid \mathbf{c} \in \mathbf{C}_{PBD} \wedge \mathbf{c}(M) > 8\}| \\ &= 36 - 3 \cdot 6 = 36 - 18 = 18 \end{aligned} \tag{2.4}$$

2.5 LEARNING MODELS WITH RANDOM FOREST

The modeling techniques considered in this paper are restricted to Random Forest (RF) regression, as the focus lies in the effect of sampling. We choose RF, as it has provided the best results in Kaltenecker et al. over all sampling strategies [46].

Table 2.13: All valid configurations $\mathbf{C}_{DB \times PBD;valid}$ resulting from the Cartesian product of distance-based sampling over the features A to G joined with the results of PBD generated for the options M, N, O

A	B	C	D	E	F	G	M	N	O
1	0	0	0	1	1	0	2	1	17
1	0	0	0	1	1	0	8	-3	7
1	0	0	0	1	1	0	8	1	1
1	0	0	0	1	1	0	32	1	7
1	0	0	0	1	1	0	2	15	7
1	0	0	0	1	1	0	32	-3	17
1	0	0	0	1	1	0	32	15	1
1	0	0	0	1	1	0	8	15	17
1	0	0	0	1	1	0	2	-3	1
1	1	1	0	1	0	1	32	1	7
1	1	1	0	1	0	1	32	-3	17
1	1	1	0	1	0	1	32	15	1
1	1	0	1	1	0	1	32	1	7
1	1	0	1	1	0	1	32	-3	17
1	1	0	1	1	0	1	32	15	1
1	1	1	1	1	0	1	32	1	7
1	1	1	1	1	0	1	32	-3	17
1	1	1	1	1	0	1	32	15	1

Random Forest (RF) is a machine-learning technique based on bagging of decision trees. In there, all trees are learned in parallel but with a varying, random feature subset to reduce the correlation between the trees. The outcome after training consists of a set of decision trees with differing internal models, e. g., concerning splitting rules, depth, and leaf nodes. The prediction for a new input is calculated as the equally-weighted mean of all tree results, which are given as the average of all values in the leaf the input ends up in. The herein used technique has been implemented in Python's scikit-learn package[3].

2.5.1 *Hyperparameters*

Many machine learning techniques offer multiple settings to control the creation of their models, parameters that do not influence the training input, but rather lay regulations on the model or the training algorithm itself. They can influence the decision making strategy inside a model, like quality criteria, or try to reduce over-fitting, e. g., via reducing the amount of iterations when learning a model. As they work on a meta level compared to the other parameters, they are often called hyperparameters [38].

For the creation of optimal prediction models, those hyperparameters need to be optimized as well [33, 74]. Therefore, one can perform a hyperparameter optimization before learning the model to find the hyperparameters best suitable for the intended purpose and data.

For RF, the implementation offers the control of multiple hyperparameters. Only a fraction of those are relevant our the execution (cf. [25, 74]), the others are just left at their default value.

- *n_estimators*: The number of estimators represents the amount of decision trees inside the forest.

3 https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, last visited at: 2022-07-23.

- *max_depth*: This can be used to regulate the maximal depth of a tree and thus reduce overfitting.
- *min_samples_leaf*: To prohibit leaf nodes with too few samples, this parameter can restrict leaves to at least the given size. Any split needs to contain at least double the amount to be considered.
- *max_features*: It restricts the number of features in the feature space for the decision trees to the given amount.
- *random_state*: This argument influences the randomness of the training, both in the choice of features to use for a decision tree, as well as in the tree training itself when determining the best split.

2.6 MODEL PROPERTIES

Multiple characteristics of a final model can be regarded to determine its quality. In the following, we present some properties that are suitable for our purpose.

MODEL PREDICTION ERROR One factor to analyze is the prediction error of the model, which is defined as the error caused by the model when applied to the whole, unsampled dataset, as used by Grebhahn, Siegmund, and Apel [33]. This property is the most obvious one, as the error is the standard criteria used to define model quality. Equation 2.5 provides the formula to calculate the Mean Absolute Percentage Error (MAPE) as used in this thesis. It is frequently used as a metric for regression models [47], since we expect small prediction errors, so roots and squares like in Root Mean Square Error (RMSE) would distort the result. Moreover, to enable a better comparison between different case studies, we use a relative error measure instead of Mean Absolute Error (MAE).

$$MAPE(\mathbf{NFP}, \widehat{\mathbf{NFP}}) = \sum_{\mathbf{c} \in \mathbf{C}_{valid}} \frac{|\mathbf{NFP}(\mathbf{c}) - \widehat{\mathbf{NFP}}(\mathbf{c})|}{\mathbf{NFP}(\mathbf{c})} \tag{2.5}$$

DURATION Another aspect to analyze is the time needed for steps in the pipeline, mainly sampling Δt_s and learning Δt_l [30]. For the sampling phase, we can observe the duration of the central aspect, which is the sampling over the different feature types according to the given strategy. Similarly, during learning, one can measure the time needed to train the model with input data. The duration for the whole pipeline Δt matches the sum for both steps ($\Delta t_s + \Delta t_l$), as those are the only phases we consider and execute in our experiment.

SAMPLE SIZE The size of the sampling result $|\mathbf{C}_{sampled}|$ is another factor to consider [33]. Although the size is not a quality indicator like the prediction error, it greatly influences the flow of the prediction pipeline. It indicates the number of times the measurements to be performed in a real setting, and also the size of the input data for the learning phase. Between those two steps, there is a trade-off, as a higher amount of samples signifies more measuring, but usually leads to a lower prediction error [26]. It can be contrasted to other metrics, or combined to e. g., analyze the throughput and cycle time.

THROUGHPUT AND CYCLE TIME The metrics duration and sample size presented above can be combined to calculate the throughput of a processing step in the pipeline, as displayed in Equation 2.6. For the sampling phase, the throughput R_s indicates the average amount of samples selected in a given time frame. Regarding the learning part, R_l shows the mean number of entries in the sample set that the specified learning technique processes to create a prediction model.

$$R_{\{s,l\}} = \frac{|\mathbf{C}_{sampled}|}{\Delta t_{\{s,l\}}} \tag{2.6}$$

Since the throughput of a phase is an objective to maximize, and the previous ones must be minimized, we invert the throughput and calculate the cycle time for the phases, as presented in Equation 2.7. The cycle time of the sampling phase T_s signifies the average duration for the addition of a single configuration to the sample set. For the learning phase, the the cycle time T_l is the average duration to process a single instance of the sample set for the prediction model. The calculation of the cycle time for the whole prediction pipeline T corresponds to the one for the pipeline duration Δt, and indicates the mean time for the pipeline to process a single configuration.

$$T_{\{s,l\}} = \frac{\Delta t_{\{s,l\}}}{|\mathbf{C}_{sampled}|} \tag{2.7}$$

FEATURE IMPORTANCE The importance of features and interactions inside the model is another conclusive factor to analyze models in-depth. Here, one evaluates the effect of a feature (or interaction) being present on the prediction. In contrast to linear models, where the influence of features and interactions can be determined directly from the formula, there are more complex mechanisms needed for tree-based strategies. Literature introduces Shapley Values to measure such an effect [59, 77], or one of their successors, like SHapley Additive exPlanations (SHAP) that combines Shapley Values with local explanatory models [55]. A variation for tree-based models, TreeSHAP, has also been published later on [54].

However, those interpretations are very complex and complicated to interpret on a global scale for the input features, and even more challenging for interactions. Consequently, we neglect this aspect for our evaluation and reduce the system to a black-box model analyzed from the outside.

2.7 STATISTICAL TESTING WITH SCOTT-KNOTT TESTS

The Scott-Knott test[4] is a statistical test capable of comparing multiple variables directly in one execution [76]. Furthermore, it is applicable for data without the assumption of a normal distribution, and for independent, unequally-sized samples. It uses the mean and variance of each variable to cluster them into different groups, which can be used to rank them later on. This test has already been used successfully in literature to group and create ranked scores for given categories [30, 84].

4 Imported as the R package *ScottKnott* [44] into the python implementation, more information is available at: https://cran.r-project.org/web/packages/ScottKnott/index.html, last visited at: 2022-09-01.

The principle of the test lies in recursively splitting a group of variables $G = \{X_1, ..., X_n\}$ by their means into two subgroups G_1, G_2 all while maximizing the sum of squares between the groups, until all possible splits are deemed insignificant [76]. The formula to calculate the sum of squares is given in Equation 2.8.

$$SS_B(G_1, G_2) = |G_1| \times \overline{G_1}^2 + |G_2| \times \overline{G_2}^2 - \overline{G}^2 \tag{2.8}$$

3 RELATED WORK

This chapter presents the results of the case study we performed to examine the usage and effect of encoding in performance prediction. As stated in the introduction, the consideration of numeric features is rare in this research area. Most publications dealing with performance analysis consider only binary features and completely exclude the existence of other option types. Thus, the following research aims to look for publications that include numeric options and find out in which form they are used to influence the prediction. This can be either directly when numeric values themselves are built into the process, or indirectly e. g., via binary encoding mechanisms pre-configured into the respective pipeline.

3.1 SYSTEMATIC LITERATURE RESEARCH

To find articles that focus on numeric features or binary encoding in performance prediction, multiple strategies from systematic literature review have been deployed. First, we searched in the papers published by the Chair[1], especially for theses and dissertations.

Afterwards, we browsed through articles published in the three most prominent conferences in Software Engineering (SE) [24], namely International Conference on Software Engineering (ICSE)[2], Automated Software Engineering (ASE)[3] and European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)[4]. Hereby, only the years 2014 to September 2022 were considered. This results in a cover of all high-rank published articles in the recent years.

Furthermore, to cover relevant articles that have not been published in those conferences, we performed a specific keyword search via combinations of the following terms:

- *(highly) configurable, software system* as the domain
- *performance prediction* as the field of research
- *performance, Non-Functional Property* as the target metric to analyze
- *sampling, learning, modeling* as the steps to perform
- *feature, configuration (option), feature / variability model* as the domain elements to perform upon
- *numeric, binary, boolean, discretize, discretization, one-hot encoding* as the applied encoding

Finally, we performed a forward and backward search for all publications identified by the means above. This way, we identify missing articles others have considered relevant for their work, or that have performed tasks similar to the ones we already named fitting.

1 https://www.se.cs.uni-saarland.de/publications.php, last visited at: 2021-11-24.
2 http://www.icse-conferences.org/, last visited at: 2022-09-17.
3 http://ase-conferences.org/, last visited at: 2022-09-12.
4 https://conf.researchr.org/series/fse, last visited , last visited at: 2022-09-15.

As restrictions for relevant publications, we require the build of performance models and the evaluation of those models on the given dataset(s). Furthermore, the analyzed software system(s) must be configurable with various parameters, which correspond to our configuration options. We do not regard configurable algorithms per se, but only systems. The approach must include numeric features, not just binary and/or categorical ones. This counts even when those features have been *one-hot encoded*. Moreover, we exclude the analysis of cloud services, since we focus on fixed software systems. Likewise, we ignore self-adaptive systems, which evaluate the performance and try to adapt at runtime. Additionally, we ignore publications in other research fields, like (performance) testing, performance regression, defect prediction, verification or compiler optimization, since those may touch our area but do not perform the required analysis.

3.2 QUANTITATIVE DESCRIPTION

Based on the literature research, 40 articles were identified that considered numeric options when analyzing performance. They are listed in Table 3.1, and presented with more detail in Table A.1 and Table A.2.

Table 3.1: Listing of all publications using numeric features and identified as relevant for this thesis

No.	Title of Publication	Authors	Year
1	"ACTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks" [7]	Bao et al.	2019
2	"An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems" [42]	Jamshidi and Casale	2016
3	"Automated Search for Configurations of Deep Neural Network Architectures" [29]	Ghamizi et al.	2019
4	"Automatic Configuration of the Cassandra Database using irace" [81]	Silva-Muñoz, Franzin, and Hugues	2021
5	"Black-Box Models for Non-Functional Properties of AI Software Systems" [27]	Friesel and Spinczyk	2022
6	"Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines" [41]	Henard et al.	2015
7	"Comparison of Analytical and Empirical Performance Models: A Case Study on Multigrid Systems" [45]	Kaltenecker	2016
8	"Data-efficient Performance Learning for Configurable Systems" [37]	Guo et al.	2018
9	"DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network" [38]	Ha and Zhang	2019
10	"Detecting Control-Flow and Performance Interactions in Highly-Configurable Systems" [20]	Denk	2017
11	"Distance-Based Sampling of Software Configuration Spaces" [47]	Kaltenecker et al.	2019
12	"Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes" [30]	Gong and Chen	2022
13	"Evolution of Performance Influences in Configurable Systems" [40]	Hasreiter	2019
14	"Experiments on Optimizing the Performance of Stencil Codes with SPL Conqueror" [31]	Grebhahn et al.	2014

No.	Title of Publication	Authors	Year
15	"Fast Performance Modeling across Different Database Versions Using Partitioned Co-Kriging" [13]	Cao et al.	2021
16	"Finding Faster Configurations Using FLASH" [65]	Nair et al.	2020
17	"Grammar-Based Sampling" [90]	Weis	2020
18	"Hdconfigor: Automatically Tuning High Dimensional Configuration Parameters for Log Search Engines" [22]	Dou, Chen, and Zheng	2020
19	"HINNPerf: Hierarchical Interaction Neural Network for Performance Prediction of Configurable Systems" [16]	Cheng, Gao, and Zheng	2022
20	"Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems" [43]	Jamshidi et al.	2018
21	"LONViZ: Unboxing the black-box of Configurable Software Systems from a Complex Networks Perspective" [52]	Li, Mao, and Chen	2022
22	"Mastering Uncertainty in Performance Estimations of Configurable Software Systems" [21]	Dorn, Apel, and Siegmund	2020
23	"Multi-Objectivizing Software Configuration Tuning" [14]	Chen and Li	2021
24	"Optimizing Performance of Stencil Code with SPL Conqueror" [34]	Grebhahn et al.	2014
25	"Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning" [78]	Shu et al.	2020
26	"Performance is not Boolean: Supporting Scalar Configuration Variables in NFP Models" [28]	Friesel and Spinczyk	2022
27	"Performance Modeling under Resource Constraints Using Deep Transfer Learning" [57]	Marathe et al.	2017
28	"Performance Prediction of Multigrid-Solver Configurations" [35]	Grebhahn et al.	2016
29	"Performance-Influence Models for Highly Configurable Systems" [79]	Siegmund et al.	2015
30	"Performance-Influence Models of Multigrid Methods: A Case Study on Triangular Grids" [32]	Grebhahn et al.	2017
31	"Predicting Performance of Software Configurations: There is no Silver Bullet" [33]	Grebhahn, Siegmund, and Apel	2019
32	"Regression Models for Performance Ranking of Configurable Systems: A Comparative Study" [15]	Chen et al.	2020
33	*Scalable Performance Models for Highly Configurable Systems* [71]	Oh and Zilmano	2020
34	"The Interplay of Compile-Time and Run-Time Options for Performance Prediction" [51]	Lesoil et al.	2021
35	"The Interplay of Sampling and Machine Learning for Software Performance Prediction" [46]	Kaltenecker et al.	2020
36	"Tradeoffs in modeling performance of highly configurable software systems" [49]	Kolesnikov et al.	2019
37	"Using Bad Learners to Find Good Configurations" [64]	Nair et al.	2017
38	"VEER: A Fast and Disagreement-Free Multi-objective Configuration Optimizer" [72]	Peng et al.	2021
39	"White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems" [88]	Velez et al.	2021
40	"White-Box Performance-Influence Models: A Profiling and Learning Approach" [89]	Weber, Apel, and Siegmund	2021

Out of those papers, 12 use or imply to use only binary encoding mechanisms to deal with numeric features. In contrast, 25 articles incorporate the numeric values directly. Only three additional publications fall in both categories. The papers applying encoding could be identified in seven cases via specifications provided in the paper itself (for instance [14, 29, 31, 40, 51]), or by the characteristics of the used case studies (e. g., [15, 21, 47]). As the set of case studies for performance predictions is very limited as of now, the origin and properties of case studies are determined easily. The majority of the publications refer to the same encoding as discussed in this work, *one-hot encoding*. However, it is sometimes referred to as *discretization*, or there is no mentioning except for the encoded case studies. But there are also three articles [20, 43, 88] encoding numeric features as binary options by leaving out some values and thus, valid configurations before starting the pipeline. Figure 3.1 displays the distribution of the input encoding.

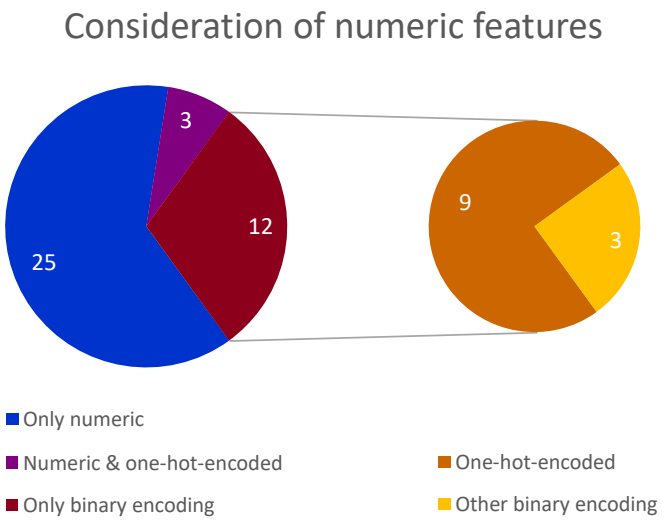


Figure 3.1: Allocation of the input methods numeric values are subjected to in various articles

Regarding the sampling techniques, for all 15 articles using binary encoding in their approaches, most use one or multiple methods presented in Section 2.4.1. A detailed list of all publications applying binary encoding is given in Table A.1. The most frequent sampling is binary random in 7 publications, followed by 6 articles with tOW sampling (with one or multiple $t \geq 2$) and 4 publications applying OW. Higher-order and hot-spot heuristic[5], distance-based as well as solver-based sampling occur in 2 articles each. Four articles also apply other techniques not mentioned previously, including iterative approaches. The distribution is visualized in Figure 3.2.

5 Similar to tOW, both are based on interactions between multiple features [34], although they don't cover all possible combinations between options.

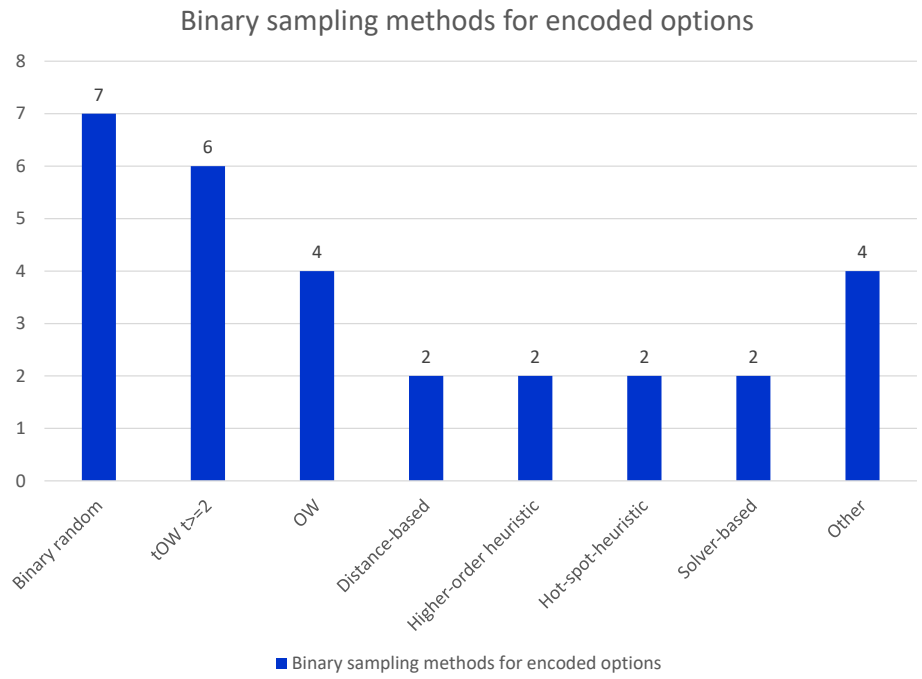


Figure 3.2: Distribution of the sampling techniques applied on binary-encoded features only

For the 28 publications considering numeric input directly, 21 implemented sampling strategies covering all configurations in one go, thereof 7 are selected via purely random sampling, and 9 applied other approaches which are partially based on a random selection of configurations. 2 articles use function learning[6]. The other 7 articles separate between binary and numeric features for sampling, and join the results via Cartesian products. Regarding the binary strategies, OW and tOW are applied in 6 articles each, followed by 2 cases of NOW as well as binary random sampling. As for the numeric techniques, most of them have been explained in Section 2.4.2. Most frequent here is PBD used in all seven articles, succeeded by 5 cases of CCID and numeric random each. Afterwards come three instances of BBD and D-Optimal Design (DOD). OFAT and FF are least deployed, with 2 respective 1 publications. The whole distribution is illustrated in Figure 3.3. All regarded publications with numeric options are listed in Table A.2.

Concerning the learning methods adopted, there is a great diversity. Most papers apply rather simple techniques like MLR [32, 34, 45, 46, 49, 79] or regression trees [37, 46, 63, 90], but a few others rely on more complex methods, like Deep Neural Networks [16, 38, 71] or Generative Adversarial Networks (GANs) [78]. The most frequently used models are learned via MLR in 15 articles (mostly in SPL Conqueror), followed by 9 cases of Classification and Regression Trees (CART). RF are applied in 6 articles, other tree-based approaches like Linear Model Trees (LMTs), XGBoost or Extra Trees occur sparsely in accordance with CART and RF. Support Vector Regression (SVR) and neural networks like Deep Feed-forward Neural Network (DFNN) appear only in 5 publications each. Other regression techniques, k-Nearest-Neighbors and Kernel-Ridge Regression (KRR) occur thrice. Four articles apply Gaussian Process Regression (GPR), mostly in accordance with Bayesian optimization, and

6 Determined polynomial influence for each combination of binary and numeric options [34].

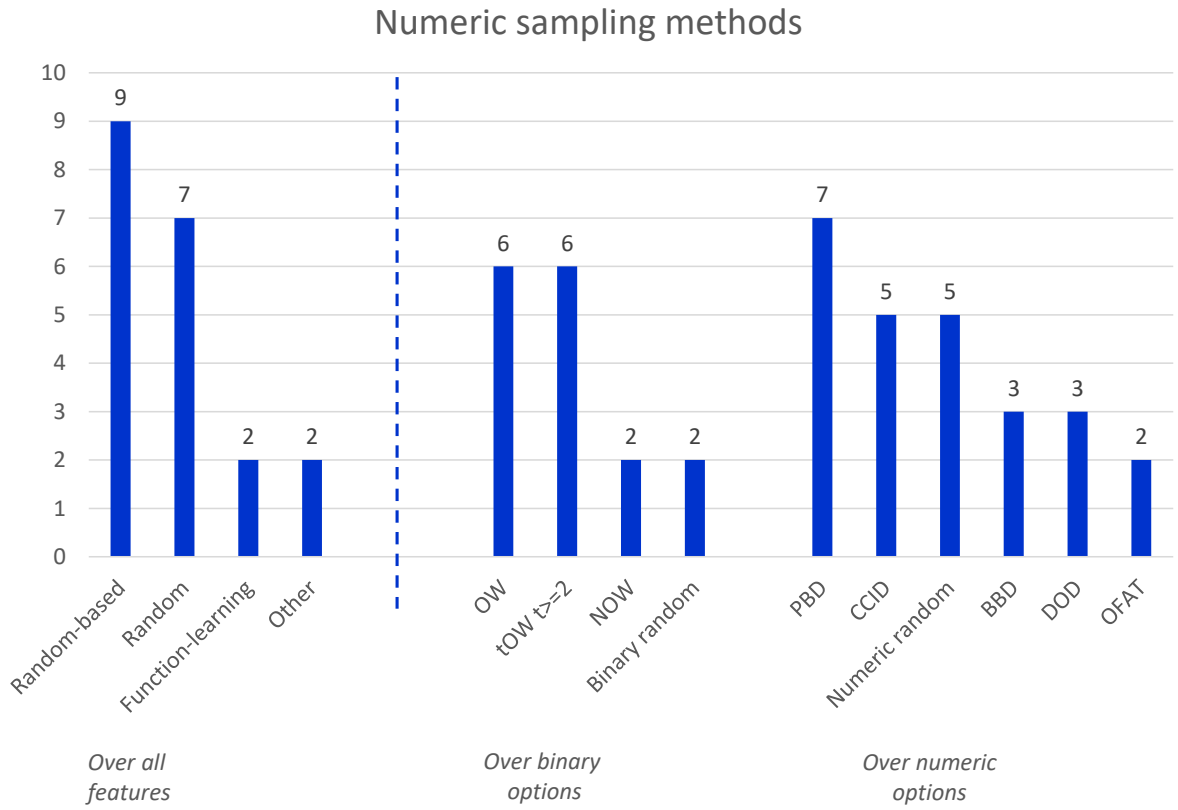


Figure 3.3: Amount of articles applying one of the named sampling strategies on numeric features To the left are the ones covering the whole feature space; whereas the middle and the right parts illustrate the strategies when splitting the configuration options per type. Binary strategies appear in the middle of the chart and numeric ones are visualized on the right side.

two apply other optimization techniques. GAN and special techniques are applied twice in the publications. A bar chart portrays this distribution in Figure 3.4.

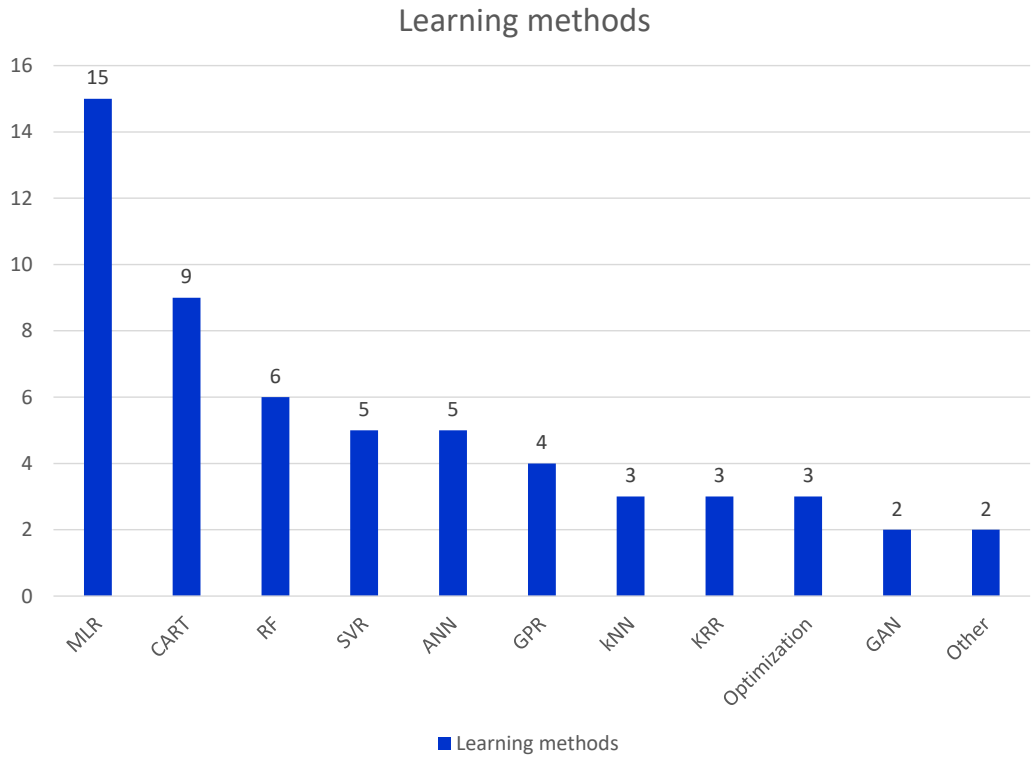


Figure 3.4: Deployment of learning techniques over identified articles

3.3 KEY FINDINGS

Even when covering this literature base, it is difficult to obtain any papers focusing on binary encoding and its effects in performance prediction. The two early articles published in 2014 by Grebhahn et al. [31, 34] tentatively apply both numeric and binary encoded pipelines. In their work, they use *discretization*, which is the same binary encoding we refer to as *one-hot encoding*. However, their analysis regarding encoding is quite insubstantial, as:

- Only few binary sampling strategies are considered in the encoded pipeline.
- Just one strategy is applied for the numeric processing, the function-learning heuristic. This technique just regards influences of all pairs of binary features mixed with numeric options. Higher-ordered feature interactions, as well as influences between just binary elements or within the numeric option space are neglected completely.
- The amount of regarded case studies is limited to only two and three systems respectively. This is too few to make a general prediction. Moreover, the results in those papers are inconclusive for the given case studies; thus no first tendencies or result expectations are recognizable.

Therefore, the outcome of the publications regarding the effect of encoding cannot be considered for the further procedure.

A recent publication by Gong and Chen is the first identified article to openly analyze and compare the encoding aspect and its effect on the performance-prediction pipeline [30]. Herein, the measurements of 5 software systems are evaluated for three encoding variants – (1) *label encoding*, which corresponds to mostly our usual numeric features, but also transforms string options into numeric features, (2) *scaled label encoding*, where the data is resized to a range of $[0;1]$ for each label encoded feature, and (3) *one-hot encoding*. However, their focus lies in the learning phase of the prediction, which is why they apply only random sampling (with 90% samples), but determine their results for seven model types: MLR, Deep Learning Model, Neural Network (NN), Decision Tree (DT), RF, k-Nearest-Neighbors Regression (kNN) and KRR. Similar to the approach utilized in this thesis, they analyze several metrics when comparing the results per encoding. Next to accuracy, which they determine via RMSE, they also regard the learning duration per encoding and ML technique as well as a trade-off between both. The evaluation is performed via a score based on the Scott-Knott test, as well as via Pareto-optima.

The results of the article show a first insight: *one-hot encoding* leads in general to a higher accuracy, although for RF, label seems to perform better. Regarding training duration, scaled label and label outperform *one-hot encoding* in this order, but for RF, label leads to the shortest training times.

Consequently, this article leads to the tentative hypothesis, that in our case numeric features might perform better than their *one-hot encoded* counterparts for both prediction accuracy and learning duration. In contrast, our work is specialized more into the sampling part of the prediction, although we also consider the overall process.

One further striking aspect when regarding all articles is that, in contrast to the possibilities displayed in Section 2.2 and Table 4.4, the chosen feature representation (binary encoding or numeric values) is always maintained for the whole pipeline.

4 IMPLEMENTATION

This chapter presents the setup of the procedure followed in this thesis. First comes the creation of artificial software systems, with feature models and configuration-based measurements. Those are detailed in Section 4.1.

Afterwards, in Section 4.2 we portray the process for choosing suitable real-life systems, which were measured beforehand.

Then, Section 4.3 describes the operations of the prediction pipeline that we need to execute for the crucial phases sampling and learning. They have been trial run with the given artificial case studies. The additional remarks about the procedure are further described in **??**. After finalizing the mechanics, the pipeline is executed on the case studies.

4.1 GENERATION OF ARTIFICIAL SOFTWARE SYSTEMS

4.1.1 *Synthetic Feature Models*

To cover various kinds of software systems, the artificial systems differentiate along the following criteria regarding the underlying feature model:

(1) The share of numeric features, how many of the options are numeric, and which ones are binary. We differentiate between considering only numeric features, just one numeric option with multiple binary ones, or a fair share of both types of features.

(2) A numeric option is also characterized by its value function in a defined range[1].:
   (a) *Linear* features imply a linear function, for instance $(2;8;+3) \rightarrow \{2;5;8\}$.
   (b) In contrast, features with *exponential* functions have the values spaced out e. g., $(4;40000;\times 100) \rightarrow \{4;400;40000\}$.
   (c) A specialty of a) are *incremental* features, like $(1;50;+1) \rightarrow \{1;2;3;...;48;49;50\}$.
   (d) *Irregular* features underlie no such basic functions, like $\{0;3;5;15;30\}$.

4.1.2 *Artificial Models for NFP Measurements*

For the NFP model imitating the measurements usually collected by running a task on a system with a tightly-constrained computing environment, we consider certain criteria:

(1) The model is constrained to linear or quadratic terms based on a single or multiple options. Other types like complex polynomial or logarithmic functions, fractions, are discounted. Example terms based on the features from Section 2.1 are:

$$3 \times C, \quad -5 \times G \times B, \quad 20 \times O, \quad 0.0003 \times M \times M, \quad -2.67185346 \times D \times N$$

1 The function for the numeric features is given as: (minimal value;maximal value;step function).

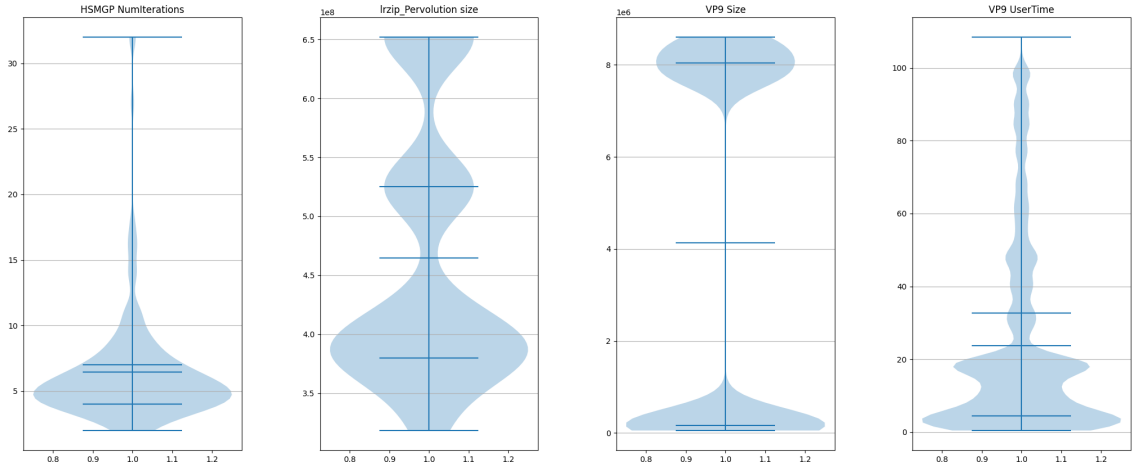
(2) The order and structure of interactions (one or multiple features involved) is crucial as well [80]. They can include just binary options, consist only of numeric features, or a mesh of both types, e. g., see the examples above.

(3) Usually the underlying model cannot be found directly via measurements, as the data may be obscured by noise of various degrees. In first instance, however, we restrict our models to clear functions without noise.

(4) The last criteria is the positivism of the model: As the example in Section 2.1.4 illustrates, the influence of features and their interactions can be either positive or negative. However, negative measurement values have little practical relevance, as the majority of performance metrics – like runtime, CPU usage, and so on – merely allow non-negative values as measurements. Therefore, to eliminate negative numbers, all non-positive values are mapped to the smallest positive entry inside the value range.

An example is a model $\mathbf{NFP} = 5 + 3 \times B - 9 \times D + 2 \times G$, which takes on the value -1 when D is *selected* and G *deselected*. The smallest positive value is 1 when both D and G are *selected*, so the resulting model is $\mathbf{NFP} = max(1; 5 + 3 \times B - 9 \times D + 2 \times G)$.

Since the system's measurements shall be as realistic as possible, we use an approach similar to the one implemented in THOR [80]. As portrayed by Siegmund, Sobernig, and Apel, the crucial aspect when generating synthetic measurements is the output distribution of NFPs. Therefore, we regard the Kernel Density Estimation (KDE) of real NFP measurements, and imitate those with artificial values. Then, we compare real and synthetic measurements manually by creating and regarding their violin plots. The following categories of real NFPs could be identified, and are exemplary visualized in Figure 4.1:



(a) Normal distribution (b) Multiple bulks (c) Dumbbell form (d) Irregular form

Figure 4.1: The four categories of measurement distributions

(a) This first category contains NFPs with approximately normal distributions. At the end of the value range, there can be a small agglomeration.

(b) The second one stations distributions with multiple bulks separated by indentations.

(c) Two focal points at each end of the value range (similar to a dumbbell form) mark the third group.

(d) More irregular forms that cannot be classified into one of the categories above are summarized within the fourth class.

4.1.3 *Presentation of Created Models*

Based on the criteria presented in Section 4.1.1, we synthesized six artificial software systems with varying feature models. Each contain zero to eleven binary options, and one to three numeric features with different value characteristics. We included incremental and exponential options, as well as some with irregularities. Their values range from a dense interval $[1;11]$ to a sparse, exponential one within $[8;32768]$. Table 4.1 presents a short overview over those systems and models.

Afterwards, we generated measurements via regression functions based on the developed feature models. They were synthesized by adapting their measurement distributions to those of existing systems. We applied mostly linear influences of features and interactions, although in a few cases we applied polynomial factors to reach certain distributions. Section A.2 in the Appendix elaborates both the feature models and the measurement distributions for each synthesized software system. Furthermore, an extended summary for all systems is introduced in Table A.3.

Table 4.1: Overview over the artificial systems generated as case studies

No.	Case Study	$\lvert\mathbf{F}_{bin}\rvert$	$\lvert\mathbf{F}_{num}\rvert$	$\lvert\mathbf{F}_{ohe}\rvert$	$\lvert\mathbf{C}_{valid}\rvert$
1	DATASET_01	3	3	21	1344
2	DATASET_02	0	3	82	6900
3	DATASET_03	5	1	12	352
4	DATASET_04	7	1	4	72
5	DATASET_05	7	1	6	80
6	DATASET_06	11	2	14	1400

4.2 SELECTION OF REAL SOFTWARE SYSTEMS

For the case studies based on real software systems, we resort to the database provided by the Chair. As the implementation of SPL Conqueror supports the splitting of feature types for sampling only in a basic setting (see Section 6.3.1), we focus on case studies with little to no cross-tree constraints. Consequently, the following case studies were ignored in our execution:

- HIPA^{cc}
- APACHE ENERGY
- APACHE PERVOLUTION
- EXASTENCILS ENERGY
- HSQLDB PERVOLUTION ENERGY
- NGINX ENERGY
- NGINX PERVOLUTION
- TRIMESH

As some of the systems were measured multiple times with different focal aspects, we reduced the amount of case studies choosing only one of those. Thus, the following case studies were disregarded as well:

- 7Z ENERGY
- LRZIP ENERGY
- MARIADB
- VP9

After the selection, we end up with 16 case studies of real software systems as presented in Table 4.2. A more detailed description of all selected systems is given in Table A.3.

Table 4.2: Overview over the real systems selected as case studies

No.	Case Study	$\lvert\mathbf{F}_{bin}\rvert$	$\lvert\mathbf{F}_{num}\rvert$	$\lvert\mathbf{F}_{ohe}\rvert$	$\lvert\mathbf{C}_{valid}\rvert$
1	7Z	10	3	33	68 640
2	BROTLI	0	2	29	180
3	DUNE	8	3	23	2 304
4	HSMGP	11	3	21	3 456
5	JAVAGC_SMALL	5	6	33	193 536
6	LIBOPUS PERVOLUTION	15	4	15	6 480
7	LRZIP PERVOLUTION	6	3	20	1 440
8	MYSQL PERVOLUTION	8	3	12	972
9	OPENVPN	18	1	5	512
10	POLLY	15	4	24	60 000
11	POSTGRESQL PERVOLUTION	5	3	12	864
12	POSTGRESQL PERVOLUTION ENERGY	5	3	12	864
13	VP8 PERVOLUTION ENERGY	9	4	17	2 736
14	VP9 PERVOLUTION	11	3	13	3 008
15	X264 PERVOLUTION	6	3	15	3 840
16	X265 PERVOLUTION	17	2	9	3 840

4.3 REALIZATION OF THE PERFORMANCE-PREDICTION PIPELINE

In contrast to the prediction pipeline presented in Section 2.2, only two phases are relevant for the conduction of our experiment: *sampling* and *learning*. One reason for this is the provision of measurement files for real software systems, which eliminates the need to measure them in the course of the experiment. Furthermore, the concrete predictions for each valid configuration are considered irrelevant for this setting. To realize the necessary steps of the prediction pipeline, each part is divided into multiple aspects.

4.3.1 *Sampling Phase*

The first part of the phase, *sampling*, begins with selecting elements from the configuration space for each feature-model variant. Thereby, we first partition the features in the feature model and then sample separately over the original binary options as well as the numeric features in their various encoding forms. Afterwards, we join the two sets via a Cartesian product. For a better understanding, Figure 4.2 presents an example for such a cut based on the *one-hot encoded* feature model portrayed in Section 2.1.3. A cut of the corresponding numeric model would partition the two features *M* and *O* off the remaining options.

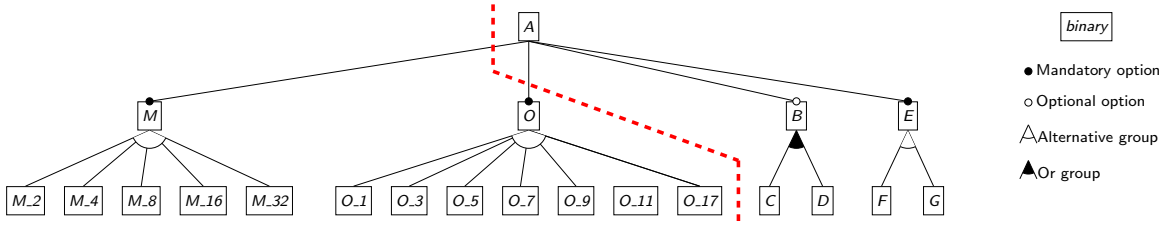


Figure 4.2: Cut of the *one-hot encoded* feature model in the sampling phase
The example is based on the feature model shown in Section 2.1.3, where option *O* is *one-hot encoded* as well, and feature *N* is left out for simplification.
The cut line separating the original binary options from the *one-hot encoded* features is marked as a red dashed line.

This procedure aids with comparing the results, as the number of samples can affect both generalization and quality of a model [26]. When contrasting pairings, an identical strategy for the original binary options reduces the difference between both techniques. For each case study, all three feature categories (original binary options, numeric-valued or *one-hot encoded* features) are granted manually a separate set of sampling strategies to choose from, depending on the properties of the feature model. Table 4.3 displays and the result set for this step in form of a systematic scheme.

Afterward, the sampled case studies are *transformed* into the oppositely encoded feature space to allow the learning of models with a different encoding than the one used for sampling (cf. Section 2.2). Thus, numeric options inside the sampled data will be *one-hot encoded*, and previously *one-hot encoded* features transformed into their numeric counterparts.

As a result, we receive four encoding categories of sampled data, which will be used in the learning phase, as displayed in Table 4.4.

Table 4.3: Scheme for the combination of sample sets to get the sampling-step result (in bold)

				Binary sampling strategy	bs_1	br_2	bs_3	bs_4	bs_5	...
				Binary result	br_1	br_2	br_3	br_4	br_5	...
One-hot encoded	Sampling strategy	ds_1	Encoded result	dr_1	$\mathbf{br_1 \times dr_1}$	$\mathbf{br_2 \times dr_1}$	$\mathbf{br_3 \times dr_1}$	$\mathbf{br_4 \times dr_1}$	$\mathbf{br_5 \times dr_1}$	...
		ds_2		dr_2	$\mathbf{br_1 \times dr_2}$	$\mathbf{br_2 \times dr_2}$	$\mathbf{br_3 \times dr_2}$	$\mathbf{br_4 \times dr_2}$	$\mathbf{br_5 \times dr_2}$	...
		ds_3		dr_3	$\mathbf{br_1 \times dr_3}$	$\mathbf{br_2 \times dr_3}$	$\mathbf{br_3 \times dr_3}$	$\mathbf{br_4 \times dr_3}$	$\mathbf{br_5 \times dr_3}$	...
		ds_4		dr_4	$\mathbf{br_1 \times dr_4}$	$\mathbf{br_2 \times dr_4}$	$\mathbf{br_3 \times dr_4}$	$\mathbf{br_4 \times dr_4}$	$\mathbf{br_5 \times dr_4}$	...
		...		...	...	...	...	...	...	...
Numeric	Sampling strategy	ns_1	Numeric result	nr_1	$\mathbf{br_1 \times nr_1}$	$\mathbf{br_2 \times nr_1}$	$\mathbf{br_3 \times nr_1}$	$\mathbf{br_4 \times nr_1}$	$\mathbf{br_5 \times nr_1}$	...
		ns_2		nr_2	$\mathbf{br_1 \times nr_2}$	$\mathbf{br_2 \times nr_2}$	$\mathbf{br_3 \times nr_2}$	$\mathbf{br_4 \times nr_2}$	$\mathbf{br_5 \times nr_2}$	...
		ns_3		nr_3	$\mathbf{br_1 \times nr_3}$	$\mathbf{br_2 \times nr_3}$	$\mathbf{br_3 \times nr_3}$	$\mathbf{br_4 \times nr_3}$	$\mathbf{br_5 \times nr_3}$	...
		ns_4		nr_4	$\mathbf{br_1 \times nr_4}$	$\mathbf{br_2 \times nr_4}$	$\mathbf{br_3 \times nr_4}$	$\mathbf{br_4 \times nr_4}$	$\mathbf{br_5 \times nr_4}$	...
		ns_5		nr_5	$\mathbf{br_1 \times nr_5}$	$\mathbf{br_2 \times nr_5}$	$\mathbf{br_3 \times nr_5}$	$\mathbf{br_4 \times nr_5}$	$\mathbf{br_5 \times nr_5}$	...
		...		...	...	...	...	...	...	...

Table 4.4: Encoding Categories in the Performance Pipeline

Prediction phase	(1) Numeric → numeric	(2) Numeric → *one-hot*	(3) *One-hot* → numeric	(4) *One-hot* → *one-hot*
Sampling	Numeric	Numeric	*One-hot encoded*	*One-hot encoded*
(Transformation)	None	*One-hot encoding*	*One-hot decoding*	None
Learning	Numeric	*One-hot encoded*	Numeric	*One-hot encoded*

4.3.2 *Learning Phase*

As machine-learning techniques like RF need to be tuned to perform optimal results [74], the second phase starts with the *learning of hyperparameters* for each case study, NFP and encoding category as portrayed in Table 4.4. Beforehand, a defined grid of possible values for those hyperparameters has been determined. At the beginning, we perform a grid search optimization, which identifies the prediction error for each accepted combination of hyper values. Thereby, it is important to consider that although the learning of the RF model takes place only with the sampled data, the final error calculation is based on the whole case study. This allows a later analysis and optimization undiluted by the sampling aspect.

Next step is the *evaluation of hyperparameters* to find the best ones within each category. As the random state may affect the results when learning, each point in the grid is executed multiple times with different seeds. Nonetheless, since the random factor is not to be analyzed in this case, the first action is to aggregate over random seeds used in the hyperparameter grid. This step is already performed inside SPL Conqueror.

Similarly, the seed-based sampling strategies – namely distance-based and random sampling – are executed multiple times with different seeds. This results in a more general

statement about those sampling strategies, as it can be highly variable. Nonetheless, the robustness of these strategies [47] suggests that the error for the samplings converges, therefore the mean prediction error of those strategies can be estimated by the mean error of several instances for a given strategy. Thus, to reduce the overhead of those strategies compared to fixed ones like, tOW, PBD or CCID, the errors of sampling strategies with identical parameters (ignoring the seeds) are aggregated. This forms the foundation for determining the optimal hyperparameters. Table 4.5 displays an exemplary calculation scheme for this process. Next, we average the results for each value combination of the hyperparameters over all sample sets. Based on this data, we identify the minimal error to decide which hyper values are to be taken. In case of equivalence regarding numbered values, we choose the minimal hyperparameter values.

The last part in this phase is the *model learning* itself, which is relying on the optimal hyperparameters determined beforehand. The results of this aspect are then used for the evaluation of the complete prediction pipeline. The output consists of analysis metrics like learning duration and prediction error.

Table 4.5: Scheme for the realization of the hyperparameter-learning phase for one ‚column' in the hyperparameter grid (parameters varying only in the seed of the random state)

Sampling strategy		Hyperp. grid	Error	Aggregate over random seeds of		
Binary[2]	Numeric	Seed	Error	1 st grid	2nd binary	3rd numeric
random(5, 15)	OFAT	32	0.21	A: 0.22	I: 0.24	1: 0.24
random(5, 15)	OFAT	43	0.23			
random(5, 22)	OFAT	32	0.27	B: 0.26		
random(5, 22)	OFAT	43	0.25			
random(7, 48)	OFAT	32	0.31	C: 0.32	II: 0.34	2: 0.34
random(7, 48)	OFAT	43	0.33			
random(7, 75)	OFAT	32	0.37	D: 0.36		
random(7, 75)	OFAT	43	0.35			
OW	OFAT	32	0.10	E: 0.21	III: 0.21	3: 0.21
OW	OFAT	43	0.32			
random(5, 15)	random(3, 67)	32	0.41	F: 0.43	IV: 0.445	4: 0.445
random(5, 15)	random(3, 67)	43	0.45			
random(5, 22)	random(3, 67)	32	0.43	G: 0.46		
random(5, 22)	random(3, 67)	43	0.49			
random(5, 15)	random(3, 81)	32	0.47	H: 0.45	V: 0.445	
random(5, 15)	random(3, 81)	43	0.43			
random(5, 22)	random(3, 81)	32	0.46	J: 0.44		
random(5, 22)	random(3, 81)	43	0.42			
random(5, 15)	random(3, 59)	32	0.48	C: 0.44	VI: 0.445	
random(5, 15)	random(3, 59)	43	0.40			
random(5, 22)	random(3, 59)	32	0.47	C: 0.45		
random(5, 22)	random(3, 59)	43	0.41			

Average error for this hyperparameter combination: 0.30875

2 The instantiation of random sampling occurs as follows: random(size, seed), where size is the number of configurations to be selected randomly, and seed is the random seed to use.

5 EXPERIMENTS

This chapter describes the main experiments that are conducted in this thesis. First, Section 5.1 presents the variables influencing the experiment and its setup including the explanation of all relevant experimental variables. In Section 5.2, we describe the research questions examined in the course of this thesis, which build the foundation for the later procedure.

5.1 EXPERIMENTAL VARIABLES

For our experiment, we identified the following variables.

The experiment and its variables are based on the data foundation to analyze; it consists of the software systems and the regarded NFPs for each system. The detailed list is given in Table A.3, including the features, configuration space and all NFPs. Additionally, a short overview has been presented in the previous chapter by Table 4.1 and Table 4.2. Based on this data, the sampling and learning steps will be performed as described in Section 4.3. The variables can be divided into four categories, by which they are presented.

INDEPENDENT VARIABLES are the varying factors in an experiment; they are selected and controlled directly by the experimenter. They are not influenced by each other or another variable [18, 75].

The central variable is formed by the encoding applied to numeric features. Those can either be left numeric or *one-hot encoded* in form of multiple binary options. For analyzing the sampling phase, only those two encoding categories are considered. Section 2.2 elaborates that the encoding can be applied both for the sampling and the learning phase independently. Therefore, when regarding the evaluation for the learning phase, we use the 4 categories presented in Table 4.4.

DEPENDENT VARIABLES describe the effect metrics to analyze. They are expected to differ based on a change in the independent factors [18, 75].

Hereby, we rely on the model properties discussed in Section 2.6. After performing the sampling phase on the data, we receive the metrics sample size $|\mathbf{C}_{sampled}|$ and sampling duration Δt_s, which can also be combined to get the sampling cycle time T_s. Next, as a result of the learning phase, we analyze the effects for the model prediction error $MAPE$, learning duration Δt_l and cycle time T_l. Furthermore, the sampling and learning durations are combined to calculate the total duration Δt and cycle time T for the prediction pipeline. Section 5.2 provides more insight into those metrics and how exactly they will be evaluated.

FIXED CONTROL VARIABLES are the factors remaining constant for each execution and throughout the whole procedure of the pipeline [17, 82]. This is, for instance, the use of SPL Conqueror for performing the sampling and learning phases. Thus, the way SPL Conqueror

performs the concrete sampling and learning tasks (for instance feature-wise sampling) has an effect on the results.

In our experiment, another fixed control variable is given by the machine-learning technique used in the learning phase of the prediction pipeline, since the technique is restricted to RF only. This choice also has an enormous influence on the metrics to analyze.

On the technical side, the hardware used for executing the pipeline is also a fixed variable. Hereby, we focus our regard for the hardware on those parts of the pipeline where we measure the sampling and learning duration, as here it influences our metrics directly.

Table 5.1: Description of the hardware performing the measured aspects in the pipeline

Phase	Computer name	CPU threads	Memory
Sampling	maxl[01-20]	1x AMD EPYC Milan 72F3	256-1024 GB RAM
		@ 3.7 GHz, 8C/16T	(limit on 250GB)
Learning	eku[04, 11]	1x Intel Core i5 − 4590	16 GB RAM
		@ 3.30 GHz 4C/4T	(limit on 3 GB)

RANDOMIZED CONTROL VARIABLES in contrast to the fixed control variables, are varied during the conduction to increases the generalization of the experiment [17, 48]. Further, the difference to independent variables is that the variation of the variable is ignored rather than analyzed. In our case, those variables usually assume values from a fixed set.

One randomized control variable poses the subset of sampling techniques chosen per case study for each feature type (original binary, numeric and *one-hot encoded*). They are selected out of a repository of sampling techniques, which remains constant throughout the whole process. Section 2.4 elaborates all sampling strategies in depth. All phases of the pipeline are executed for each combination of techniques in the sets. The concrete effect of each strategy is ignored in the overall analysis, as only the aggregated version of the metrics is considered for each category and case study.

Random states during sampling or (hyperparameter) learning are also randomized control variables, as the steps are executed repeatedly with a fixed set of seeds, but their results are analyzed in an aggregated form to increase the robustness of the execution [47].

The optimal hyperparameters for the learning phase form another control variable, which are evaluated after the sampling. The regarded grid used for the hyperparameter optimization with RF remains constant, but the concrete optimum may differ given the scenario. The chosen hyperparameters (ignoring the seed) are permitted to vary across the independent variable, as well as case study and NFP. Section 4.3.2 describes the optimization procedure in more detail, and Table 5.2 displays the grid of accepted hyperparameters, which are elaborated in Section 2.5.1. In contrast to the other randomized variables, we do not perform the learning phase multiple times with different values for the hyperparameters and aggregate the results. Rather we only determine the overall optimal combination, and execute the learning step with those parameters (ignoring the seed).

Table 5.2: Listing of all hyperparameters allowed to vary during the execution, and their range of accepted values

No.	Hyperparameter	Value range	Values used in grid search
1	max_depth	int	{100000000; 5; 10; 20}
2	max_features	{"auto", "sqrt", "log2"}, int or float	{"*auto*"; "*sqrt*"; "*log*2"}
3	min_samples_leaf	int or float	{0.001; 0.003; 0.005}
4	n_estimators	int	{100; 200; 400}
5	random_state	int, RandomState instance or None	{790554120; 989443602; 426026004; 297788243; 642376861; 96118863; 56621629; 621404434}

5.2 RESEARCH QUESTIONS

In this section, we present the research questions posed in the given experiment, whose variable setup is described above.

5.2.1 *Sample Size*

RQ0: How do sample sizes differ depending on the encoding of numeric options across all case studies?

Here we regard whether the chosen encoding has an influence on the size of the sample sets resulting from the sampling phase. Some of the upcoming research questions include the sample size in their analysis, therefore this question is the foundation for evaluating the research questions and must be regarded beforehand. The query itself does not represent a research question, as the result cannot be used to draw a conclusion on the quality of a system. Nonetheless, it can be used as a cost metric, since in real-life scenarios, the sample size indicates the amount of configurations to measure and thus the effort to take before analyzing a software system.

RESTRICTION The analysis includes only the sampling strategies which have fixed-size results depending on the case study's feature model. Varying techniques like distance-based or random sampling are exempt from this question, as their size is predetermined by the user before execution.

VALIDITY The sample size of some strategies depends on the implementation inside SPL Conqueror, thus another program might come to differing results. Nonetheless, the experimental designs usually have a fixed implementation and thus, their difference should be minimal.

REALIZATION As an evaluation metric we use the rank-based score introduced in Section 5.3 with the metric *sample size* $|\mathbf{C}_{sampled}|$ across case studies and sampling strategies for the encoding categories *numeric* versus *one-hot encoded*. Because of the symmetrical sampling setup for both encoding strategies and the minimum of cross-tree constraints inside the feature models, one binary sampling strategy is exemplary and can represent all others. Therefore, we adapt the *binary aggregation* step of the calculation by determining the sample sizes for one binary sampling strategy only instead of aggregating them. The rest of the procedure is performed as described.

5.2.2 *Sampling Duration*

RQ1: What is the effect of encoding numeric options on the duration of sampling across all case studies?

To our knowledge, there has been no analysis comparing the sampling duration depending on the encoding of numeric features inside the feature model. We would like to evaluate whether the time span needed to sample over the configuration space differs depending on the encoding category (*one-hot encoded* or numeric).

RESTRICTION The measured duration includes only the time span needed for the sampling procedures. Consequently, the loading of the feature model and measurement files, the check for validity of all measurements, as well as the combination of sampling results and their storage in a file are not taken into account. This leads to a better comparability between case studies, as the fluctuating amount of entries inside the measurements could distort the measure by adding enormous overheads.

VALIDITY The sampling duration depends highly on the performance of SPL Conqueror, thus another implementation might come to completely different results. Furthermore, there is no guarantee that SPL Conqueror has been optimized for all sampling strategies, so the differences in performance may (partially) also be caused due to unequally performing implementations.

REALIZATION The calculation of the evaluation metric is performed as elaborated in Section 5.3. As a metric, we apply *sampling duration* Δt_s and *sampling cycle time* T_s, as described in Section 2.6, across case studies and sampling strategies for the encoding categories *numeric* versus *one-hot encoded*.

5.2.3 *Learning and Total Duration*

RQ2: What influence has the encoding of numeric options on the duration of learning, and the performance prediction in total across all case studies?

Evaluations of Machine Learning (ML) executions in different branches have detected that *one-hot encoding* increases the complexity of the data and thus the learning duration [4]. First outcomes show that the surge in training time applies also for learning performance models [30], especially for RFs. We would like to evaluate whether this effect takes place in

our situation as well, and how significant it is. For the sampling phase, there was no such statement found, thus we have no prior expectations for the previous question.

Furthermore, since we regard the encoding during the whole pipeline, and sampling and learning can have their encoding independent of each other, we receive 4 learning categories to differentiate for this question (cf. Table 4.4).

REALIZATION For this research question, we calculate the evaluation metric as detailed in Section 5.3. However, we now apply four metrics: *learning duration* Δt_l, *learning cycle time* T_l, *total duration* Δt and *total cycle time* T. As categories, we now regard the four encoding categories described in the independent variables paragraph of Section 5.1. They are evaluated across case studies, NFP and sampling strategies.

5.2.4 *Prediction Error*

RQ3: How does the encoding of numeric options during sampling and machine-learning affect the error of performance prediction across all case studies?

As of now, the effect of encoding during sampling on the prediction error is hardly researched.

Based on the knowledge of our team, we were able to formulate some hypotheses about the differences this encoding causes, which may have an effect on the prediction error. For instance, there may be a loss of information due to a change in value conception. *One-hot encoding* regards the values of a feature independent of each other, and draws no assumptions or relations between those. Therefore, Bao et al. suggests it for categorical options [7]. With regard to numeric encoding, it might be helpful for features whose influence on the prediction model is irregular. In contrast, numeric features allow the use of functions over the feature space, which can simplify the prediction model and deliver more precise answers even for values not covered by sampling results [33]. However, since RF does not learn a global function, this advantage will not show in our scenario.

Furthermore, *one-hot encoding* reduces the potential influence of each value feature in the configuration space, since a numeric option with v values causes each *one-hot encoded* value feature to appear in just about $\frac{1}{v}$ cases compared to the original numeric feature. This may affect splitting rules or limitations.

Recently, a first article appeared, which discussed the effect of encoding on prediction errors in more detail [30]. However, the focus lies on learning effects of encoding, and the sampling is always performed randomly with a 90% share of samples, which cannot be considered sampling in a practical scenario at all. Therefore, we concentrate on the sampling aspect of the encoding, and restrict the learning technique to RF only. Nonetheless, it is impossible to ignore the machine learning aspect completely, thus it will also appear in our analysis.

REALIZATION The evaluation metric, elaborated in Section 5.3, is calculated in accordance with the last question Section 5.2.3. The regarded metric is now the *model prediction error MAPE* issued by SPL Conqueror as described in Section 2.6.

5.3 EVALUATION METRIC

For evaluating and analyzing the research questions, we calculate a rank-based evaluation score. This procedure is applied to the data provided by the prediction pipeline for each research question. The evaluation metric is based on a rank-based score for each encoding category across all sampling strategies and case studies. For the questions relating to data provided by the learning phase, we also include the NFP into the dimensions to create the score over. This is indicated by the round brackets () in the step descriptions below. The procedure for calculating the rank-score is as follows for each measured metric:

(0) *Available data*: As a result of the execution of the prediction pipeline, we receive a value for any case study (and NFP for **RQ2** and **RQ3**) and for each combination of strategies applied to the case study. To reduce the overhead of sampling strategies executed multiple times with varying seeds, the values are aggregated prior to evaluation. This happens in analogy to the aggregation procedure for hyperparameter evaluation in Table 4.5, so that we receive a mean value for all strategies that use seeds, as given in the last column of the table.

Thus, we receive the following data for all case studies (and NFP): For each binary and each encoded sampling strategy used in the case study (ignoring the seed), there is a value representing the measured metric by encoding category.

(1) *Binary aggregation*: Since the binary sampling strategies are not targeted in this evaluation, we begin by calculating the mean value across all binary sampling strategies for each case study, (every NFP) and encoding category.

(2) *Numeric aggregation and Grouping*: Next, we aggregate over the numeric respective *one-hot encoded* sampling strategies. A simple aggregation would have been to calculate the mean for each encoding category. As the data given here is not distributed normally, but rather irregular with dense clusters and sparse areas, just using the mean would lead to a high loss of information.

In contrast, statistical tests that do not require normally-distributed data, like the Mann-Whitney U or the Wilcoxon signed-rank test [56, 91], usually rely on ranks for their analysis. However, a simple ranking of means over this irregular data would distort the results greatly as well, e. g., for the means $\{0.1, 0.2, 50\}$ of each category this could lead to a ranking of $\{1, 2, 3\}$, where the difference of one rank is on one hand caused by a delta of 0.1, and on the other hand by a delta of 49.8. Therefore, it does not make sense to apply ranks directly on means for the categories.

Instead, we group similarly-distributed instances of the measured metric across all encoding categories for each case study (and NFP) before ranking them. For given example, this would result in a grouping $\{1, 1, 2\}$, which is less distorted. Another advantage hereby is that we can base our grouping and ranking not just on the simple mean, but also include other factors like the variance. The grouping is done via the Scott-Knott test based on the mean and variance of the numeric respective *one-hot encoded* sampling strategies for each case study (and NFP). Hereby, a smaller group index implies a smaller mean value over all encoding strategies referenced, and this is preferable for all regarded metrics. As a result, each case study, (NFP) and encoding

category has been assigned a group representing the metric across the sampling strategies for the numeric features.

(3) *Category ranking*: Those resulting groups are then ranked across the encoding categories for each case study (and NFP). As a ranking strategy, we choose fractional ranking, since it is used in the aforementioned statistical tests [56, 91], and preserves the sum over all ranks. Those ranks take values between 1 and the product of the number of encoding categories with the amount of case studies (and NFP), which is in our case $2 \times 22 = 44$ ($4 \times 64 = 256$). Hereby a smaller group signifies once again a smaller rank, and thus a better result. For the given example, this means a ranking of $\{1.5, 1.5, 3\}$.

(4) *Case study aggregation*: Now we can aggregate the categories over all case studies (and NFPs) by summing up the ranks for each encoding category. Since those sums increase quadratically by the amount of entries, we divide it by the total rank sum. Thus, we receive our final score for each encoding category, and their sum makes up 1. Those rank-based scores can now be interpreted by their difference to 0.5 (0.25) and each other, thereby a smaller score implies a better result.

Figure 5.1 visualizes the scheme for this realization on the metric *sampling duration* for the two encoding categories numeric and *one-hot encoding*, and additionally provides an example.

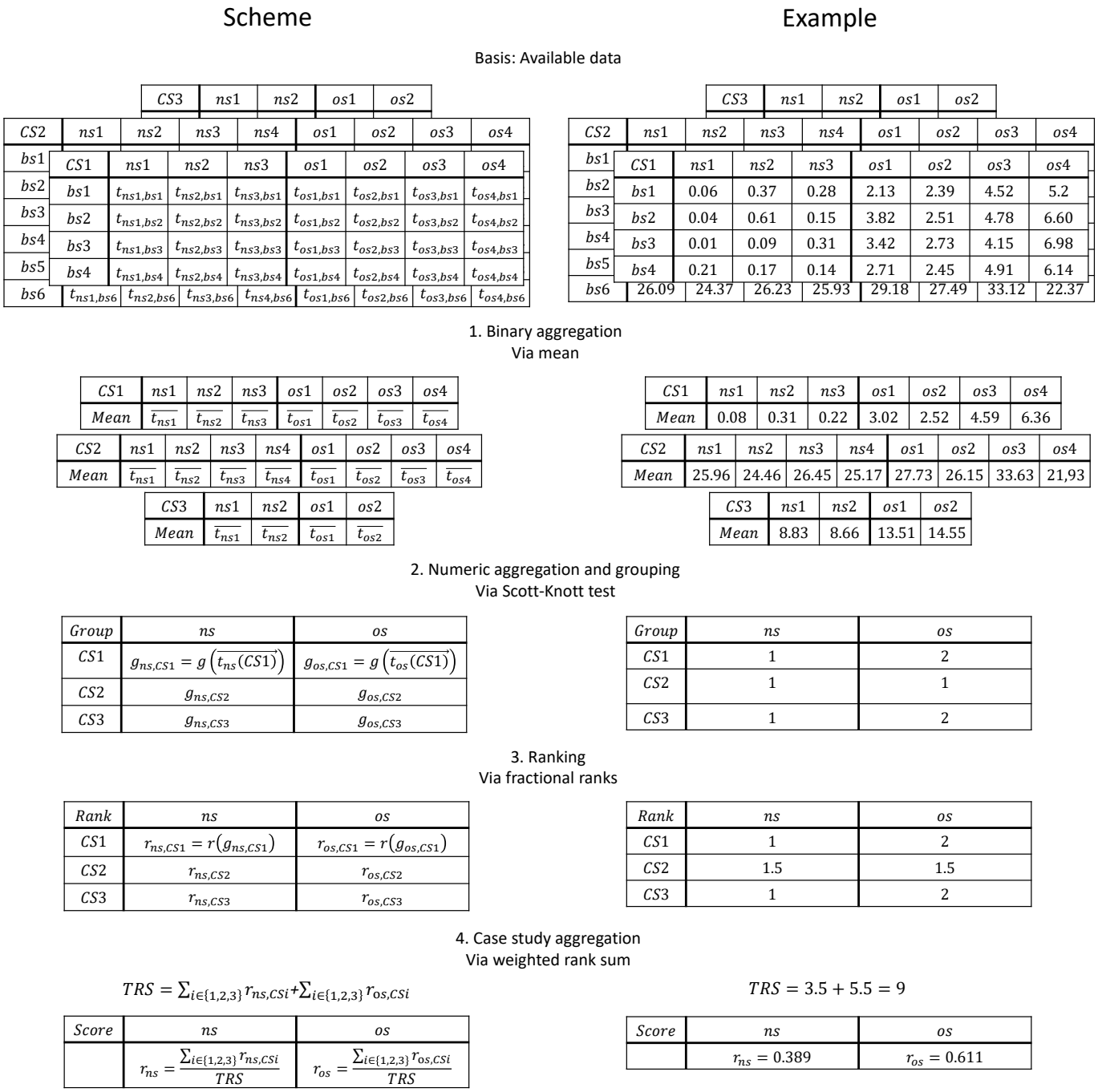


Figure 5.1: Scheme and example for calculating the rank-based evaluation score on the metric *sampling duration* for the two encoding categories *numeric* and *one-hot encoding*

6 EVALUATION

In this chapter, we perform the analysis pertaining the experiment and the pipeline execution described in the previous chapters. In Section 6.1, we evaluate the results of the experiment described in the previous chapter. The next section, Section 6.2 gives more context about the evaluated metrics as well as the optimal hyperparameters used for learning. Finally in Section 6.3, we describe the constraints placed on the execution and experiment that may affect the results portrayed in Section 6.1.

6.1 RESULTS

In this section, we present and evaluate the results of the research questions portrayed in Section 5.2. Table 6.1 summarizes the evaluation scores for each research question.

Table 6.1: Summarized results of the evaluation score for all research questions
Behind each score, we added a rank in brackets to accentuate the order of the categories per metric. The best scores for each metric are highlighted in **bold**.
Hint: Due to round-off errors, the sum of all scores may not match 1 for each metric.

	Metric	Categories			
	Sampling	Numeric		*One-hot*	
rq_i	*Learning*	Numeric	*One-hot*	Numeric	*One-hot*
0	Sample size ($\lvert\mathbf{C}_{sampled}\rvert$)	**0.4318** (1)		0.5682 (2)	
1	Sampling duration (Δt_s)	**0.4848** (1)		0.5152 (2)	
1	Sampling cycle time (T_s)	**0.5000** (1.5)		**0.5000** (1.5)	
2	Learning duration (Δt_l)	**0.2055** (1)	0.2313 (3)	0.2289 (2)	0.3344 (4)
2	Learning cycle time (T_l)	0.2875 (3)	0.3000 (4)	**0.2031** (1)	0.2094 (2)
2	Duration (Δt)	**0.2148** (1)	0.2367 (2)	0.2609 (3)	0.2875 (4)
2	Cycle time (T)	0.2711 (3)	0.2773 (4)	**0.2242** (1)	0.2273 (2)
3	Prediction error ($MAPE$)	0.2813 (4)	0.2242 (2)	0.2766 (3)	**0.2180** (1)

SAMPLE SIZE Here, we receive a score of 0.4318 for the numeric and 0.5682 for the *one-hot encoded* sampling. This indicates that the sample size is much smaller for sampling strategies based on numeric features compared to the use of *one-hot encoding* for fixed-size strategies.

RQ0: The sample size is much smaller when treating numeric features in the sampling phase than when applying *one-hot encoding*.

SAMPLING DURATION Concerning the sampling duration, we get similar results for the two encoding strategies (0.4848 and 0.5152, respectively). However, the difference between

the scores is much less, which signifies a lesser rank distance between the categories. In contrast, the scores for the sampling cycle time are identical with 0.5000 for both, so we cannot determine a rank-based difference between numeric and *one-hot encoding*.

RQ1: The duration of the sampling phase is lower for sampling over numeric than *one-hot encoded* features. Regarding the sampling cycle time, we detected no significant difference.

LEARNING AND TOTAL DURATION The learning duration shows three rather similar categories, and the fourth one, *one-hot encoded* category. With 0.3344, *one-hot encoding* has the highest score and thus, it is much worse than the others. The numeric category has the lowest score with 0.2055, afterwards comes the *one-hot encoded* sampling 0.2289 and nearby the *one-hot encoded* learning 0.2313. This indicates that learning with *one-hot encoding* results in a larger learning duration, which may be due to the higher dimensional complexity. *One-hot encoded* sampling also increases the duration, which may stem from the larger sample size. Both reasons combined might explain the high score for the *one-hot encoded* category.

Concerning the learning cycle time, there is a clear divide between numeric and *one-hot encoded* sampling of $8 - 10\%$, where the last categories perform more efficiently with 0.2031 and 0.2094 for numeric respective *one-hot encoded* learning. Numeric sampling leads to a score of 0.2875 versus 0.3000. The higher score for *one-hot encoded* sampling can be a result of the larger sample size.

For the pipeline's duration we see results combining the properties of sampling and learning duration. Thus, the numeric category performs best with the lowest score of 0.2148, followed by their *one-hot encoded* learning counterpart with 0.2367 and its reversal 0.2609. The *one-hot encoded* category brings up the rear with 0.2875. The larger divide between the second and third-ranked categories is due to the difference in sampling.

Similarly to the learning one, the cycle time of the whole pipeline has similar scores for the categories. Due to the sampling influence, they are only on a tighter scale. *One-hot encoded* sampling has the scores 0.2242 and 0.2273, and 0.2711 respective 0.2773 for numeric versus *one-hot encoded* learning.

RQ2: The learning duration differs depending on the encoding applied in the sampling and learning phases of performance prediction. The best result is obtained by an all-numeric pipeline. It is followed by *one-hot encoded* sampling combined with numeric learning, and close-by its reversal. Finally, the learning phase lasts considerably longer with *one-hot encoded* prediction.
For the learning cycle time, the encoding in the sampling phase is crucial, and in the learning step quite negligible. Hereby, one-hot encoded sampling performs a lot more efficiently compared to the numeric counterpart, and numeric learning leads to slightly better results per sampling strategy.
The duration of the prediction pipeline is uniformly distributed with regards to the encoding categories. Similar to the duration of both sampling and learning steps, sampling over numeric features is significantly faster than categories with *one-hot encoded* sampling. Likewise, within each sampling encoding, the categories based on numeric learning perform quicker as well.
The cycle time of the pipeline performs in analogy to the learning cycle time.

PREDICTION ERROR For this metric, we can see a huge distinction in the scores of the four categories regarding the learning encoding, whereas between the sampling encoding there

is only a slight difference. With a score of 0.2180, the *one-hot* category has a minimal rank-based error across all case studies and NFP, and the corresponding category with numeric sampling is a close second with 0.2242. Afterward comes the numeric learning categories with 0.2766 respective 0.2813 for *one-hot encoding* versus numeric sampling. Consequently, the encoding during learning has an enormous effect on the prediction error, the sampling's feature encoding only very little. Regarding the small descent in score stemming from the sampling encoding, it might correlate with the increase in sample size.

RQ3: For the prediction error, we can detect crucial differences regarding the encoding categories. In contrast, the output now depends mostly on the encoding applied in the learning phase. Learning with *one-hot encoded* features performs considerably better than with numeric options. Within those phases, *one-hot encoded* sampling achieves slightly better predictions than its numeric counterpart.

6.2 DISCUSSION

6.2.1 *Analysis of Selected Hyperparameters*

For the hyperparameter analysis and evaluation performed as described in Section 4.3.2, we use a common strategy of k-cross validation ($k = 5$) in combination with a grid search algorithm [1, 16, 30]. Thereby, we can identify the prediction errors across all points of the grid portrayed in Table 5.2. Afterwards, the results for each combination of hyperparameters are aggregated separately and the one with minimal error is selected.

Bergstra and Bengio suggest random search instead of grid search for an efficient optimization [9], which has been applied multiple times in literature for performance prediction [10, 33, 46]. Nonetheless, the grid search algorithm is more practical and feasible in our setup, because of aggregation inside the evaluation. When using the grid search, we can just aggregate over the predefined grid points in the hyperparameter dimensions to find the optimal one. However, we may miss global or even local minima with grid search if they do not coincide with the given grid points. In contrast, when using random search, one would have to estimate an error function over the dimensions for each combination of sampling strategies before the aggregation and then resume by calculating the global minima over the aggregated error function.

Here the danger of missing minima is reduced with random search and a good approximation of the error functions. However this approach increases the amount of computation immensely. For our prediction, the accuracy of grid search optimization suffices.

6.2.2 *Distribution of Dependent Variables*

In addition to the evaluation scores presented in Section 6.1, we can also regard the distributions for each metric, across all case studies and sampling strategies. They are plotted across all categories as well as for each category separately. This can give an orientation of the value range the metrics lie in, and allows us to contrast the distributions per category against each other. Thus, we can also get a better understanding of the research questions. As our procedure of calculating the evaluation metric is not a standard approach,

we can use the insights gained from those plots to verify our score results portrayed in Table 6.1.

After regarding the distribution plots portrayed in Figure 6.1, we can see a high span of outliers that dominate all plots, and that the major bulk of values lies in the lower 10% of the value range, in most cases it is even less.

Therefore, we additionally regard the distribution of the metrics also on a logarithmic scale, as Figure 6.2 illustrates. Based on those charts, we can make multiple observations:

(a) For the *sample size*, the *one-hot encoded* samples are usually larger than the numeric ones, as the distribution is identical with an offset for the *one-hot encoded* sample sizes. The high amount of samples is sometimes quite large, with up to 60 000 samples in the outliers. In a real setting, those are definitely not wanted, since measuring NFP for thousands of configurations is rather impractical.

Thus, we intuitively expect a higher ranking for the *one-hot encoded* than the numeric category, as stated in the results.

(b) Regarding the *sampling duration*, the metrics look alike but the numeric values more densely distributed, and both start and end of the value range lie lower than those of *one-hot encoded* sampling duration. Most of the values lie below 10 s and thus are rather irrelevant in the pipeline execution, however the outlier indicate the period of half an hour in an unfortunate setting.

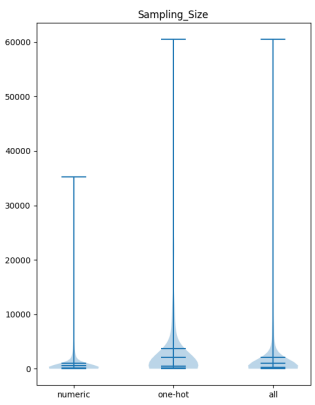
Consequently, we should receive a higher ranking for the *one-hot encoded* than the numeric category as well, which is fulfilled by the results.

(c) Concerning the *sampling cycle time*, the distributions are reversed. Both bulk and outlier range end earlier for the encoded than the numeric cycle time. However, the start of the *one-hot encoded* value range lies above the numeric one. This difference between duration and cycle time has its reasons in the higher sampling size, e. g.,the runs for binary-random sampling can issue a high amount of samples corresponding to a tOW strategy in very little time.

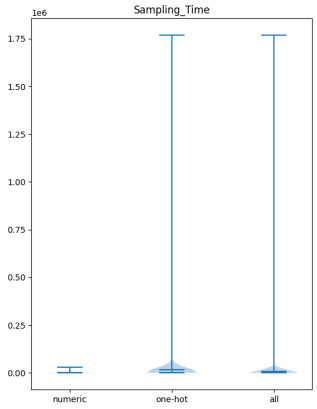
For this metric, it is difficult to determine ranking orders manually. Here, the results postulate no significant difference.

(d) The four categories of the *learning duration* differ as well. Thereby, the duration for categories with numeric sampling use a smaller value range and upper focal point than those starting with *one-hot encoded* sampling. Between the distributions of the numeric-sampled categories, there is a large difference in the shape of the focal range. Nonetheless, it is difficult to interpret those differences and rank one category above another. In contrast, the categories starting with *one-hot encoded* sampling show a larger difference, since *one-hot encoded* learning results in a more widespread distribution with a slightly higher starting point than the distribution performed with numeric learning. This indicates, that *one-hot encoded* sampling and learning perform much longer than those with numeric learning. Most of the values lie below 1 s, but can go up to half a minute, and thus can be ignored even more.

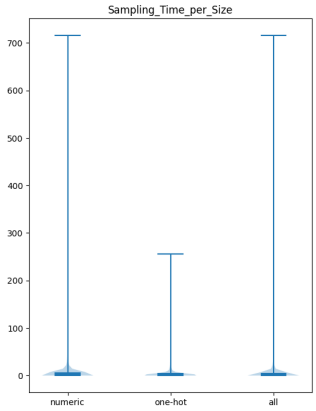
As a result, we should receive the lowest ranking for the numeric sampling categories, together with the *one-hot encoded* sampling and numeric learning. The *one-hot encoded*



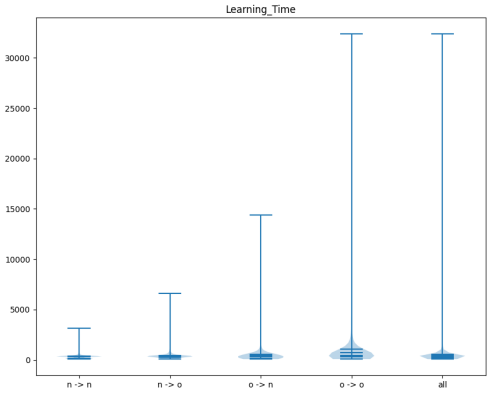
(a) Sample Size



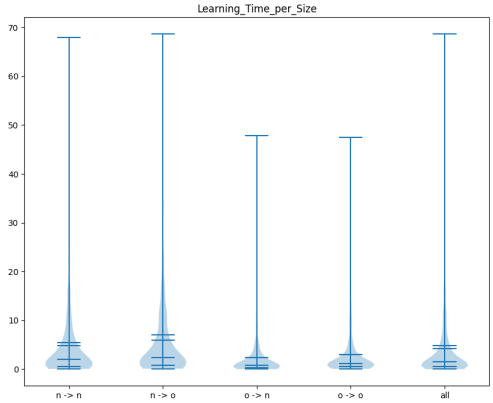
(b) Sampling Duration



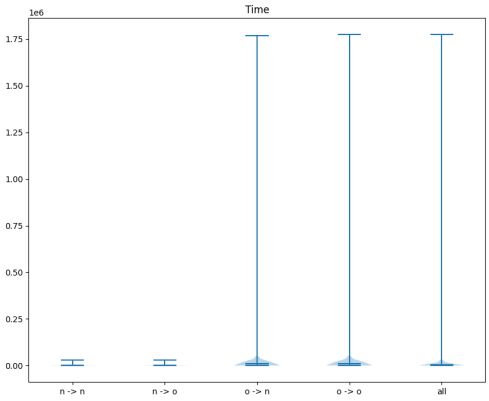
(c) Sampling Cycle Time



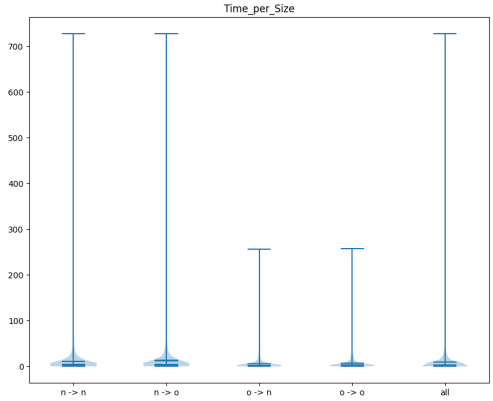
(d) Learning Duration



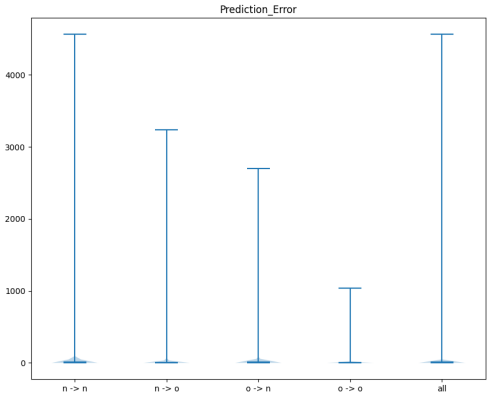
(e) Learning Cycle Time



(f) Total Duration



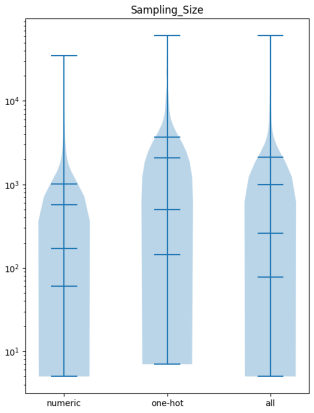
(g) Total Cycle Time



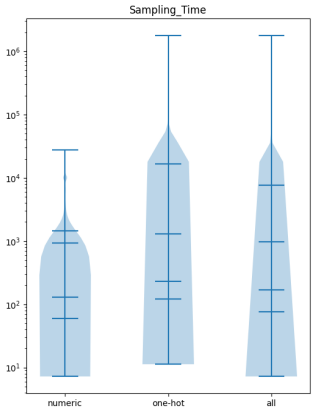
(h) Prediction Error

Legend:
Regarding the category names in the plot, ‚n' stands for numeric and ‚o' for *one-hot encoded*. The letter in front of the arrow indicates the feature representation at the sampling phase, the letter behind stand for the encoding used for learning. When different, there was a transformation between those steps.

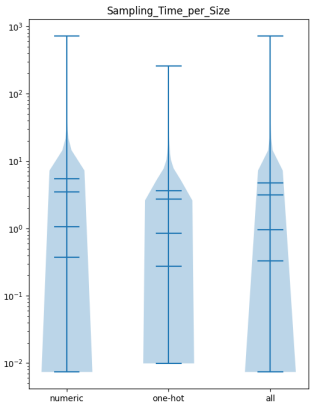
Figure 6.1: Distributions of metrics all together and per category, on a linear scale



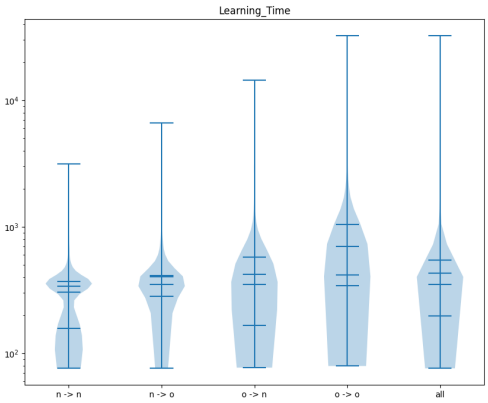
(a) Sample Size



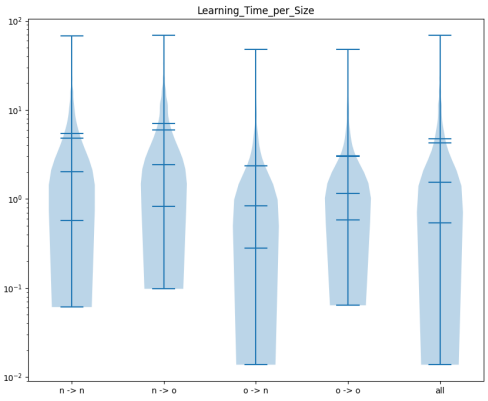
(b) Sampling Duration



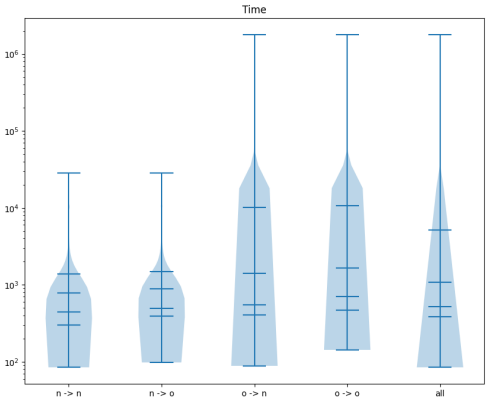
(c) Sampling Cycle Time



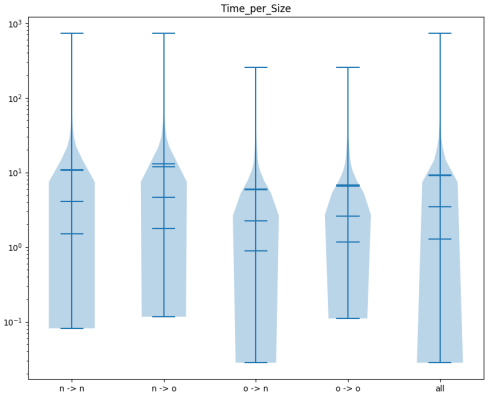
(d) Learning Duration



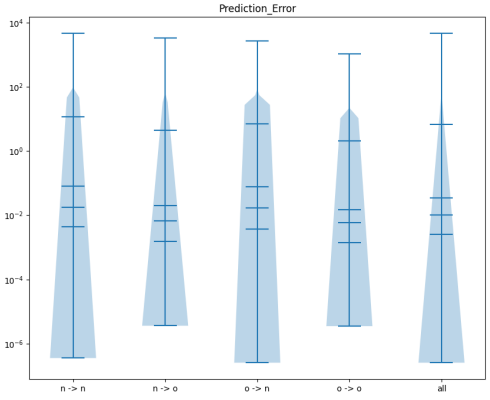
(e) Learning Cycle Time



(f) Total Duration



(g) Total Cycle Time



(h) Prediction Error

Legend:
Regarding the category names in the plot, ‚n' stands for numeric and ‚o' for *one-hot encoded*. The letter in front of the arrow indicates the feature representation at the sampling phase, the letter behind stand for the encoding used for learning. When different, there was a transformation between those steps.

Figure 6.2: Distribution of metrics all together and per category, on a logarithmic scale

category should achieve the highest score by far. This is given by the result, although numeric sampling and *one-hot encoded* learning scores higher than its reversal.

(e) Concerning the *learning cycle time*, the results differ much from the corresponding duration measurement. One category is highlighted, *one-hot encoded* sampling combined with numeric learning, since it results in smaller values than they can be detected in other categories. The upper range however looks similar to the category with *one-hot encoded* learning. Similarly but on a smaller scale, the same counts for the categories of numeric sampling. when comparing the total numeric respective *one-hot encoded* categories, *one-hot encoded* performs better as the upper distribution is lower than the one for numeric.

Thus, we intuitively expect a higher ranking for the numeric sampling and *one-hot encoded* learning, followed by the numeric and afterward *one-hot encoded* category. *One-hot encoded* sampling with numeric learning brings up the rear. This also validates the results, although the delta in score is higher between the different types of sampling encoding.

(f) For the *total duration*, the distribution ranges (with and without outliers) for the categories are plotted similar to the sampling duration. Both distributions of the categories sampling over numeric features encompass a smaller value range, and less distant outliers. Out of both, the category with *one-hot encoded* learning leads to a denser bulk, as the range starts at a higher value. In a similar way but more widely spread upwards, the categories with *one-hot encoded* sampling are analogous, here the range with *one-hot encoded* learning includes a higher starting point. In analogy to the sampling duration, which on average make up a higher part of the total duration, most of the values lie below 10 s. Nonetheless, the outlier indicate the duration of half an hour in an unfortunate setting.

Consequently, we should receive the best (minimal) scores for the numeric sampling categories, followed by *one-hot encoded* sampling combined with numeric learning and the *one-hot encoded* category. The results reflect those ideas, although they also portray the same distance between numeric and *one-hot encoded* learning for numeric sampling.

(g) Regarding the *total cycle time*, the distributions are analogous to those of the learning cycle time, although on a different scale. The only major difference is between the *one-hot encoded* learning categories, as both distributions start at about the same value.

Therefore, we expect a higher but similar ranking for the numeric sampling categories, followed by afterward *one-hot encoded* category and the *one-hot encoded* sampling with numeric learning. This is fulfilled by the results, with a significant delta between the sampling categories.

(h) In contrast to the size, duration and cycle time metrics, the distributions for the *prediction error* are formed less like a goth window, but rather they fade upwards uniformly. Furthermore, the range starts at a much lower value, at less than 10^{-4}. However, the interpretation of the value range above 10^{0} is quite disappointing, since the values indicate a relative error of up to 100 in the bulk and 5 000 for the outliers,

which signifies a very poor model quality in certain instances of case study, NFP and sampling strategies. When contrasting the distributions, it can be noticed that the *one-hot encoded* learning categories both start on a much higher value than those with numeric learning, although those value differences are negligible in reality. The ranges' and bulks' ends are all on about the same level, but the categories with *one-hot encoded* learning are represented less in the upper regions. For the numeric sampling, this is portrayed by a much more pointed top, and for *one-hot encoded* sampling the top is lower and also more pointed than the totally *one-hot encoded* execution.

As a result, we should receive the highest score for the numeric learning categories, followed by learning with *one-hot encoding*. In between the sampling encoding, we expect an increase in score from *one-hot encoded* to numeric sampling. This is reflected by the evaluation scores, although the difference between the learning categories is much higher than expected, and overshadows the small delta between the sampling strategies.

When regarding the distribution of the dependent variables, it can be noticed that all metrics cover a huge interval on the value axis. The major part of the span is taken by outliers that stem from single instances of a few case studies only, the main share of the values lie in the lower 10% of the range.

Based on the metrics' distributions provided in form of the plots, we could verify the results for each metric as presented by the evaluation scores in Section 6.1.

6.3 THREATS TO VALIDITY

During the conduction of this experiment, we made assumptions and restrictions that threaten the results presented in Section 6.1. Thereby, we differentiate between internal validity, which is affected by decisions that shape the internal stability of an experiment, and external validity, which is influenced by limitations affecting the generalization of the outcome.

6.3.1 *Internal Validity*

To get comparable results when sampling, the sampling of binary features and *one-hot encoded* options is split and their results are merged via a Cartesian product. This enables the use of sampling strategies for both feature types separately, and therefore we can cover a larger base of sample sets with *one-hot encoding*, as all strategies chosen for binary and *one-hot encoded* features can be combined. The same procedure is mandatory for numeric features, thus this technique is used for a better comparison of the results and the chosen subsets for numeric respective *one-hot encoded* options. Despite that, we have to consider that the merged sample set usually is larger than when applying the sampling strategy to the whole feature set.

The implementation of SPL Conqueror supports the use of cross-tree constraints in sampling only for whole feature type subsets, thus constraints are only considered when sampling over either all binary options or all numeric features at once. For our execution, when sampling binary and *one-hot encoded* options separately, this can result in invalid

configurations being forwarded. To deter those, the final results (after Cartesian product) are all tested for validity once again. Nonetheless, the partial results of a sampling may already be invalid due to those constraints, which has not been tested beforehand. In contrast, in most binary strategies, the recursive model constraints are taken into account. Thus, for those a major part of the invalid feature space is already forbidden. Therefore, this procedure of checking constraints at the end can reduce the sample set significantly.

To maintain the size of a sampling set even under cross-tree constraints, a solver could find the closest valid configuration. An alternative approach would be to consider the constraints in sampling strategies where at a certain point in time, the strategy must decide between multiple configurations with an equal heuristic. Those techniques, however, are not considered in this thesis, as sampling completion is not the focal aspect and they would complicate the setting even further. Because of those two flaws in execution, we restricted the choice of case studies to those with none or minimal cross-tree constraints, such that the sampling of potentially invalid configurations is minimized.

For each feature representation, we chose only one single strategy to sample over their sub configuration space. In a real setting, it might make sense to combine multiple strategies to reduce the weaknesses of each approach, e. g., PW with BBD for all numeric features.

As elaborated in the discussion, the hyperparameter learning and evaluation is performed with grid search and an aggregation of the errors across configurations. Consequently, we do not identify the global optimum over all combination of hyperparameters, but only the minimum of all grid points. The alternative is calculating the error functions for each configuration, and then determining the global minimum over the aggregation of those functions. This adds a colossal calculation overhead, therefore we decided on the grid search as a hyperparameter learning strategy.

The hardware during the execution of the measured sampling and learning aspects of the pipeline is kept constant by machines with identical hardware and set up, as described in Table 5.1. This guarantees the comparability of the resulting metrics.

The evaluation metric works on a rank-based score with significant grouping of means, which can easily be distorted by the results of a single sampling strategy. Especially the first mean over the binary sampling strategies can make an enormous difference to the analyzed metric.

6.3.2 *External Validity*

To achieve a sufficient overview of the state-of the art in the given topic, multiple systematic strategies for literature review were applied (cf. Chapter 3). Moreover, it has also not been cited by any publication of those categories, and therefore it is seen as negligible. Recent articles published after September 2022 will be disregarded as well. We might have missed some publications while performing snowballing if they have not been cited in any of the publications.

For the results to be as general as possible, various real-world configurable systems were analyzed. In addition, synthesized but realistic case studies of varying size and feature space were examined to broaden the perspective.

In the feature models constructed for all systems, we pre-encoded the categorical options via *one-hot encoding*, as the encoding preserves the nominal characteristic and does not

assume an ordering. This is performed similar to the works of Bao et al. [7], and in contrast to the (scaled) label encoding used by Gong and Chen [30].

The measurements taken for the real software systems are not based on the complete feature model of the system including every single option, as this would be infeasible for most systems. For example, regard the Linux kernel with more than 12 000 options in 2022 [1]. Based on a subset of 8 700 of those features, over 95 000 random configurations took more than 15 000 hours to measure the build of the system. Instead, most of our models consist of the main features and the others are set to default when measuring. For some numeric options, their value range was also reduced to a commonly-applied range [92]. Both restrictions sample over the system options before even starting the performance-prediction pipeline, so this pre-sampling reduces the explainability for the whole system.

For generating the synthesized case studies, we do not employ the application THOR [80], since it cannot include numeric features. However, we incorporated its approach, the consideration of measurement distributions when modeling NFPs. Thereby, we identified four categories of distributions in real case studies and imitated those with linear and polynomial models.

Furthermore, the procedure for calculating the evaluation metric takes into account multiple sampling strategies for the binary features and the (encoded) numeric options over which we aggregate the results, thus we reach a higher generalization.

During the procedure of the performance-prediction pipeline, there are multiple components inside the sampling and learning phase which are based on (pseudo)-random influence. For those, multiple tries with different random seeds have been established. This includes the strategies for random and distance-based sampling, which have been executed with two to five different, randomly chosen seeds. In the hyperparameter learning, the grid contains also 8 fixed random states, over which the mean prediction error is optimized. During the hyperparameter evaluation, the errors are averaged over those seeds to receive more general results. Only afterwards the best hyperparameter setting for all sample sets is chosen. Similarly, the evaluation score calculated for the research questions aggregates over the given random seeds in the sampling strategies before calculating the final score.

Moreover, the split in the feature domain for sampling is not common in the area of performance-prediction and thus less generally applicable.

The restriction of learning methods to RF reduces the generalization, as the results cannot be transferred to other techniques, especially to ones based on a completely different learning concept like MLR or neural networks.

7 CONCLUDING REMARKS

7.1 CONCLUSION

Many articles analyzing performance prediction in software systems encode numeric features explicitly or implicitly before applying their pipeline, even though they take part in almost all software systems. Despite that, the effect of encoding in this field has been sparsely researched. To fill that void, we contrast the effect of *one-hot encoding* versus numeric feature representation. Thereby, we focus on the sampling aspect of performance prediction, but also consider the influence of the learning phase with regards to sample size, step duration and prediction error.

Our results demonstrate that the encoding definitely has an enormous, non-negligible effect on the performance prediction. The sample size and sampling duration are both significantly smaller when using numeric-encoded feature representations. Regarding the corresponding cycle time, there is no significant evidence for an encoding effect. Also for the learning phase and the whole prediction pipeline, the duration is minimal for numeric sampling and learning, and maximal for a wholly *one-hot encoded* execution. All in all, the sampling duration makes up a larger share of the pipeline than the learning with RF, and both times are negligible, as the major portion lasts less than 10 seconds during execution. Due to the higher sample size, the (learning) cycle time is optimal for *one-hot encoded* sampling and numeric learning. Concerning the most relevant metric, the model prediction error, the effect of encoding is crucial as well. Here, the best scores are reached with *one-hot encoded* learning, and optimal also with the same encoding in sampling. Therefore, this thesis provides a deeper insight into the effect of encoding numeric features and paves the way for further research in this area. The effect could be analyzed per sampling strategy in-depth, or for other learning techniques to receive a more general impression.

7.2 FUTURE WORK

The results presented in this thesis have been retrieved from a higher amount of case studies compared to other approaches and publications in the field of performance prediction [30, 34, 38, 61]. Nonetheless, concerning the vast amount of software systems in existence, it is still a minuscule amount. Thus, the conclusion could be solidified on a higher number of case studies to receive a more legit answer. The valid configuration space of the regarded software systems are rather small compared to systems in real life. Therefore, a validation with larger systems might be favorable as well.

Moreover, we manipulated the sampling process for the *one-hot encoded* feature variants, since we split up the feature domain and sampled on binary and *one-hot encoded* options separately. Consequently, one might want to perform this research again with a wholesome sampling approach on the encoded feature models.

Furthermore, we fixated the encoding for the whole domain of numeric features, however, the optimal feature representation could also vary between options. This could get evaluated in another experiment.

Additionally, we restricted the sampling strategies to those given in SPL Conqueror. Nonetheless, there exist other sampling strategies that are disregarded in this application, like t-wise interaction sampling [3, 50]. Regarding the encoding techniques applied, many more are available which have not been included in this thesis, like *scaled label encoding* [30] or alternative binary encoding techniques mentioned in Section 2.1.3.

Finally, we compared the encoding strategies as a whole group in each scenario. It might be interesting to contrast the performance of all included strategies as well to receive a more detailed outcome and create a sampling guideline.

APPENDIX

A.1 EXTENSION OF THE QUANTITATIVE DESCRIPTION

This section elaborates further quantitative aspects about the literature research, which are not as crucial for the performance of this thesis, but may of interest to the public.

The chronological distribution of the 40 publication years ranges from 2014 to 2022. At the first phase, from 2014 to 2018, there was little interest into the topic, and only 2 to 3 publications were issued annually. An exception is 2017 with five submissions. Later on, the research increased, as starting from 2019, at least five articles with numeric input were published. The peak is situated in 2019 and 2021 with 7 publications in both years. Figure A.1 displays the chronological publication of the articles.

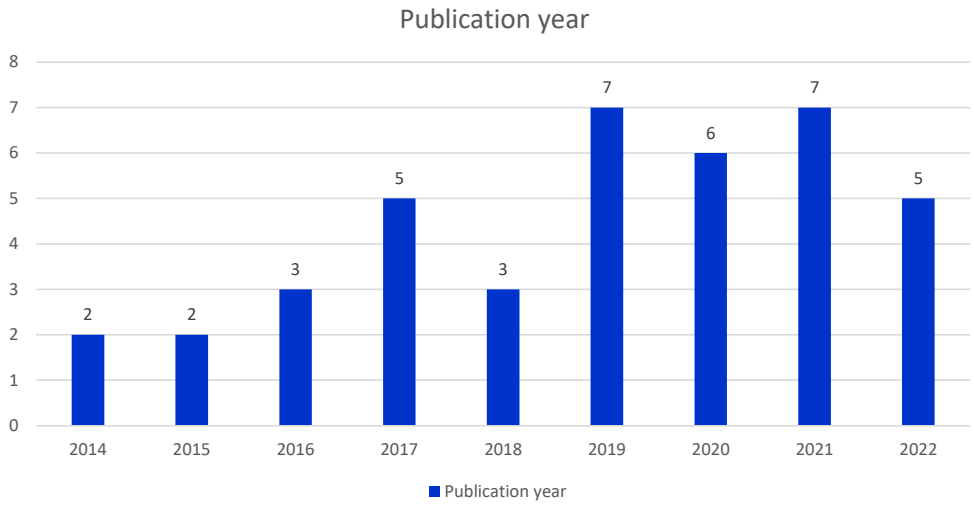


Figure A.1: Distribution of the publication year for the articles

Regarding the origin of the articles, the entries are from mostly Europe, but also America, Asia and Australia. Out of 31 articles created within Europe, 23 stem from Germany, and only 5 have been contributed to by the UK. Luxembourg, Belgium and France added 2 respective 1 publication each. Then, North America follows with 11 publications, of which the US has taken part in 10 articles and Canada one. Asia (China) contributed to 8 publications and 2 originated in the country Australia. Figure A.2 portrays the origin of all publishers.

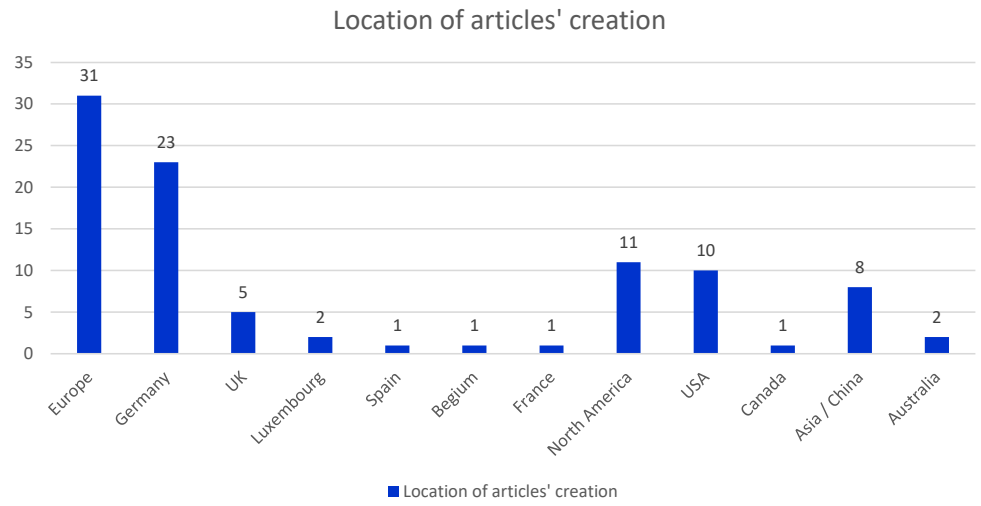


Figure A.2: The origin country of all publishers involved in an article
As multiple parties from different countries may cooperate for some articles, the overall sum as well as the sums in each continent may not add up.

The Chair of SE has contributed to 20 articles (independent of the Chair's location at the time of publishing); only half of all publications were created without relation to the Chair. This may be partly due to a selection bias, as the articles and theses originating from the Chair are easily accessible. However, the Chair has also invested a great deal of research into the topic, and cooperated with many other institutes to publish articles together. This is also visualized in Figure A.3.

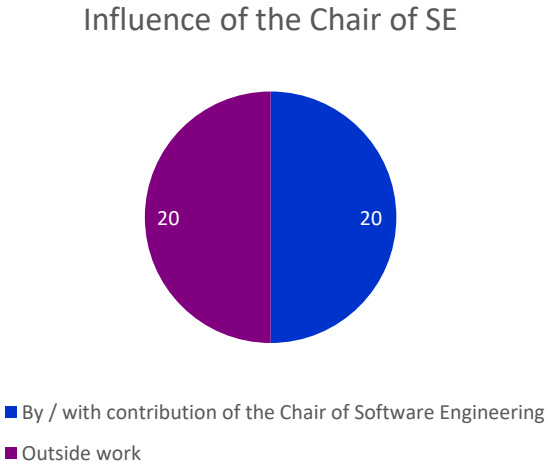


Figure A.3: The influence of the Chair of SE on the selected articles

A.2 ARTIFICIAL SYSTEM MODELS

This section presents the six artificial software systems that were generated for this thesis. They are characterized by their feature models, and their synthesized measurements imitating the NFPs of real systems. For each system, we will display the corresponding feature diagram. The measurements will be provided in form of violin plots visualizing their distribution. A short summary of the systems is given in Table 4.1 and Table A.3.

DATASET_01 has, with three each, a fair share of both binary and numeric features. It includes 2 linear numeric options with 7 and 8 values, as well as an exponential one with just 3 values. The feature diagram is depicted in Figure A.4.

Afterwards, we generated and selected four measurements. *Linear_model_06* pertains to category (a), as it has a normal distribution with upward outliers. The second one, *linear_model_07*, with a dumbbell form, is part of category (c). Similar to the first measurement, *linear_model_09* is also a member of category (a), although with less severe outliers that form a small bulk at top range. Category (b) is represented by *linear_model_11*, with multiple agglomerations at different parts of the value domain. Figure A.5 illustrates their value distributions.

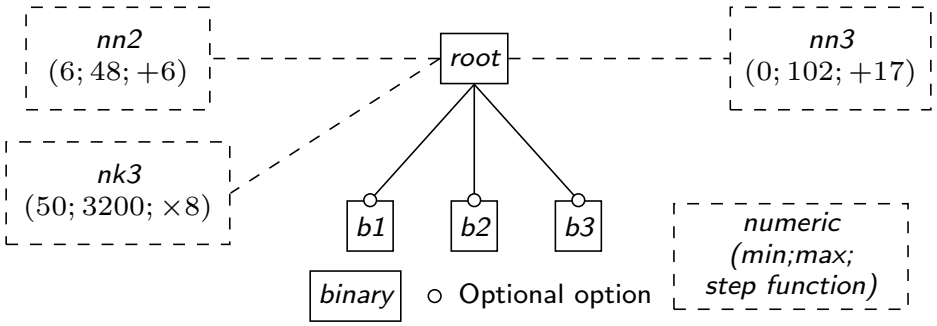


Figure A.4: Feature diagram for artificial DATASET_01

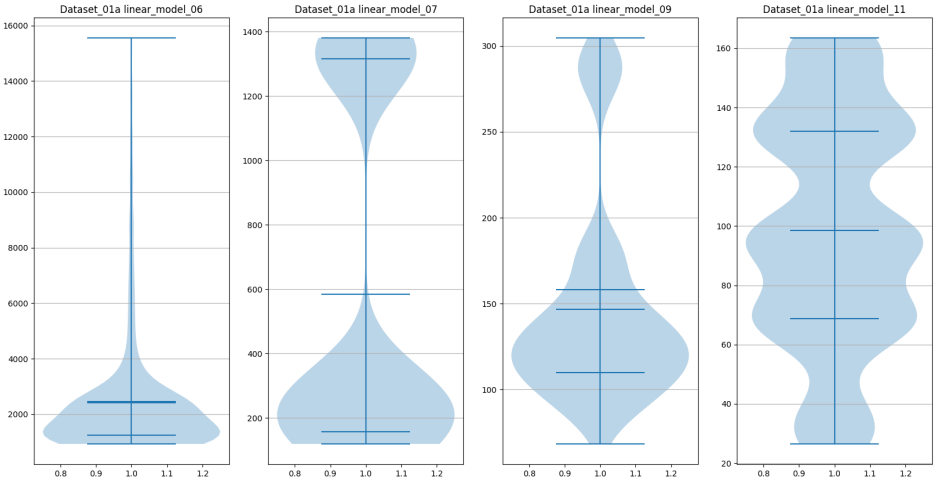


Figure A.5: Four measurement distributions for artificial DATASET_01

DATASET_02 does not include any binary options, but consists of three linear numeric features. Two of those include many values, the other one consists of much less values that extend to up to 1 800. Its feature diagram is visualized in Figure A.6.

We administered four NFPs. The *linear_model_02* is an abnormal one, bulking steeply at zero and with a high lance upwards. Thus, it is assigned to category (d). *Linear_model_04* and *linear_model_07* both belong to category (a). Whereas the *linear_model_04* has few outliers and a flattened normal distribution, the *linear_model_07* resembles a rhombus with outliers upwards. Finally, there is again a member of category (b), *linear_model_10*, with multiple, unevenly spaced agglomerations. Figure A.7 presents the plots for the measurements.

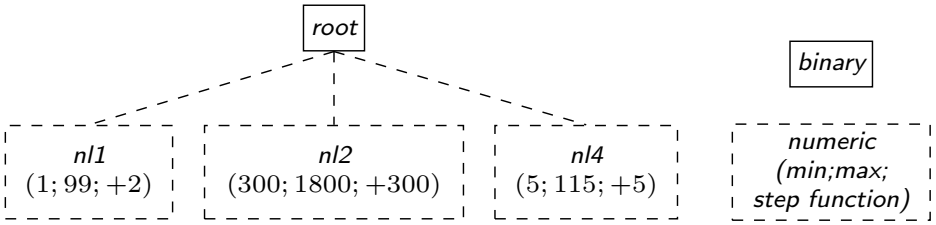


Figure A.6: Feature diagram for artificial DATASET_02

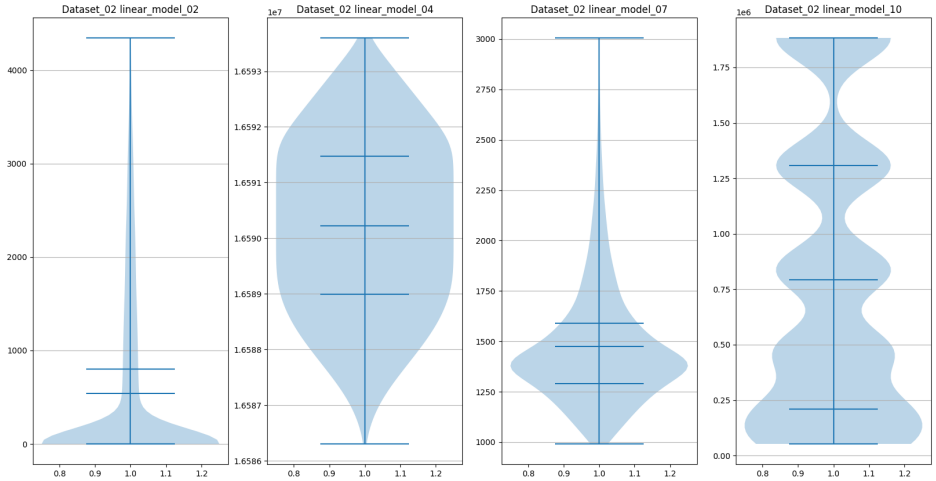


Figure A.7: Measurement distributions for artificial DATASET_02

DATASET_03 contains a single, dense incremental numeric option in the range $[1;11]$. Additionally, we added five binary features, that are optional and completely independent of each other. The feature diagram is portrayed in Figure A.8.

Once again, we synthesized four measurements suiting the system. Similar to the one above we have two members of category (a), *linear_model_02* flattened, and *linear_model_06* steeper, with small bulky outliers. *Linear_model_04* has multiple agglomerations, and is a part of category (b), whereas *linear_model_05* is a dumbbell of category (c). Figure A.9 visualizes the measurement distributions.

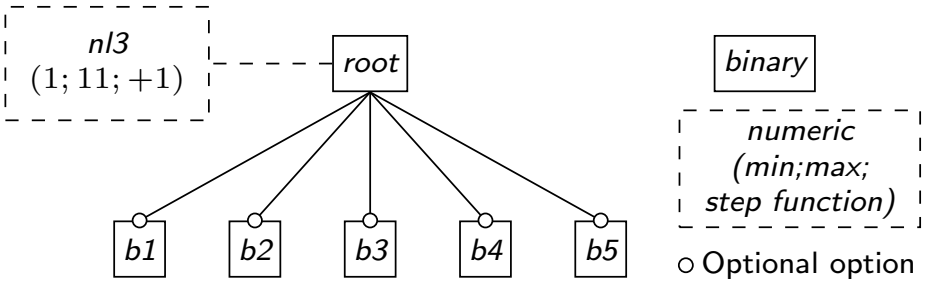


Figure A.8: Feature diagram for artificial DATASET_03

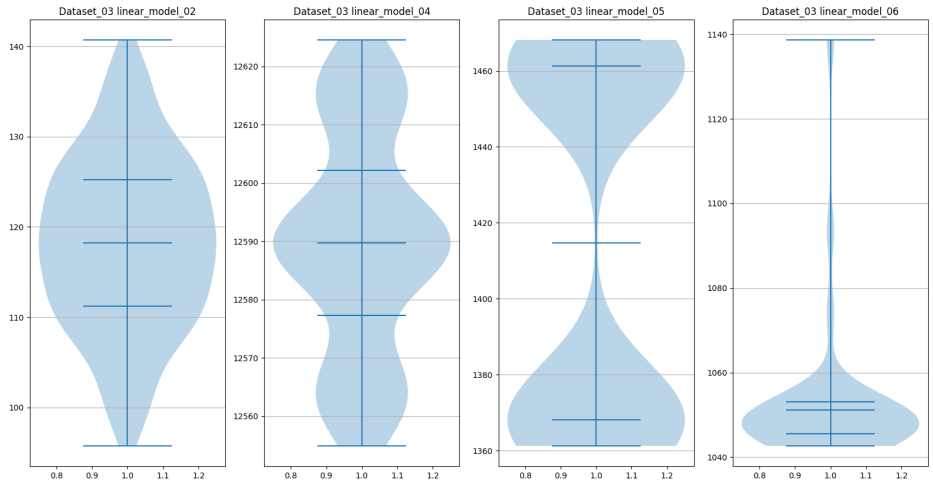


Figure A.9: Measurement distributions for artificial DATASET_03

DATASET_04 Figure A.10 displays the feature diagram of the fourth artificial system, which contains an irregular numeric option with only three values. In addition, it includes seven binary features, of which one is an optional child of another, and two form an alternative group to yet another parent.

In contrast to the systems portrayed above, this dataset includes only three generated measurements displayed in Figure A.11. First, we have a model of category (c), *linear_model_01*, which has the shape of a dumbbell. *Linear_model_04*, in contrast, is more irregular and thus part of category (d). Category (a) is represented by *linear_model_06*.

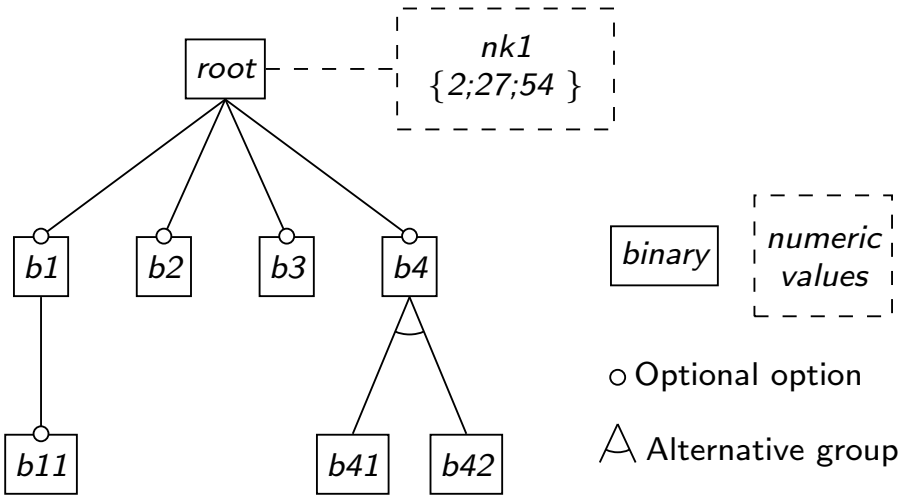


Figure A.10: Feature diagram for artificial DATASET_04

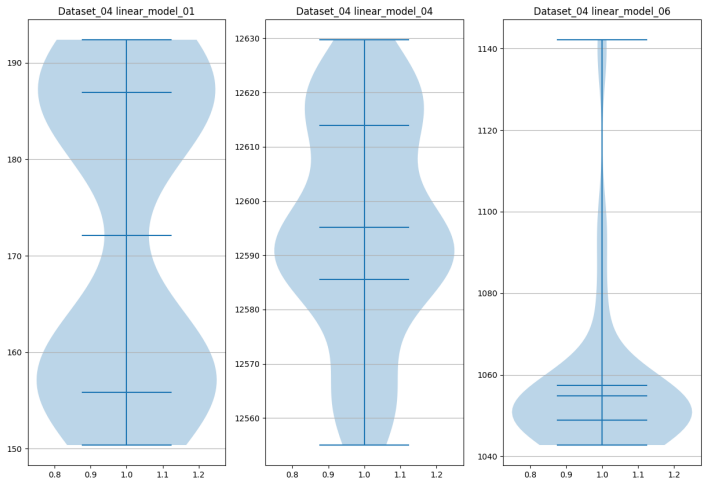


Figure A.11: Measurement distributions for artificial DATASET_04

DATASET_05 also includes just a single numeric feature, this time however a linear option in $[0;48]$ with a missing value in-between. Furthermore, it encompasses again seven binary features, and three form an alternative group under another optional feature. The diagram is displayed in Figure A.12.

In accordance with the model, we generated three synthesized NFPs, illustrated in Figure A.13. *Linear_model_02* is a part of category (a), with a normal distribution and no outliers. Category (c) is represented by *linear_model_03*, and the more irregular *linear_model_05* is a member of category (d).

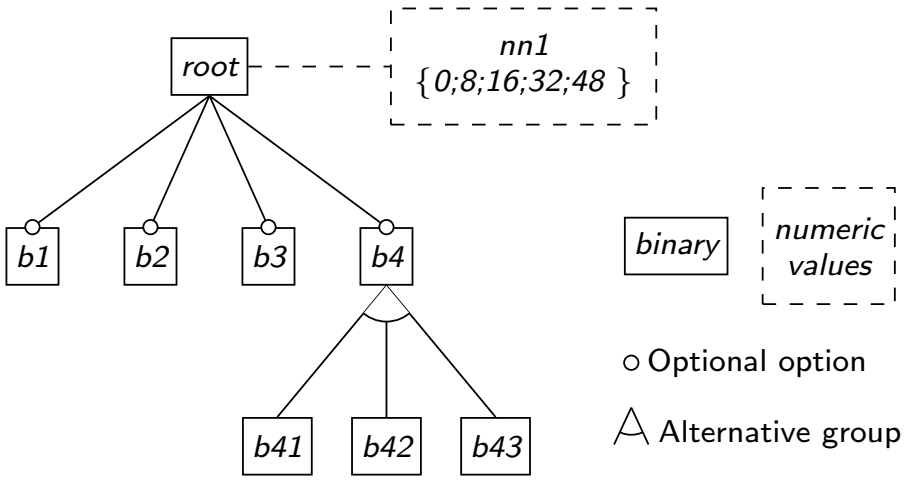


Figure A.12: Feature diagram for artificial DATASET_05

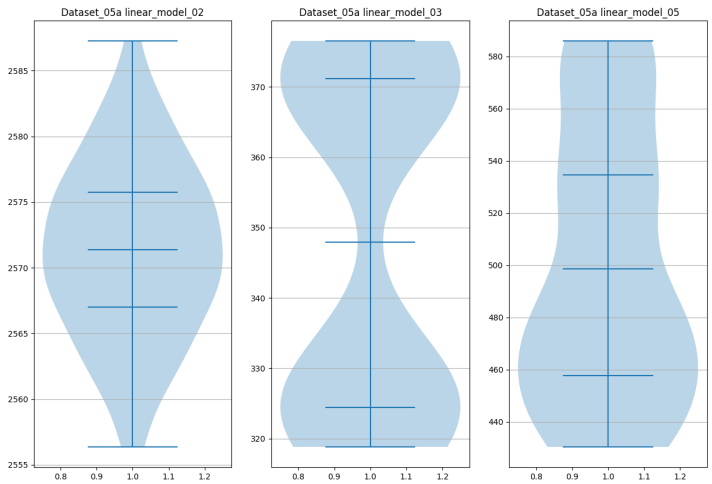


Figure A.13: Measurement distributions for artificial DATASET_05

DATASET_06 includes two numeric features, in addition to eleven hierarchically-organized binary options. The numeric ones are both exponential ones, once on base 2 and the other on base 8. For the binary ones, we have two alternative groups of size two and five, each under an optional binary parent. The entire diagram is depicted in Figure A.14.

Regarding the measurement distributions, we have synthesized three NFPs, as illustrated in Figure A.15. *Linear_model_02* has multiple agglomerations, and is a member of category (b). In contrast, *linear_model_03* resembles more a normal distribution, however with some outliers. Thus, it belongs to category (a). At last, *linear_model_04* pertains to category (c), as it has a dumbbell form, even though there are some outliers.

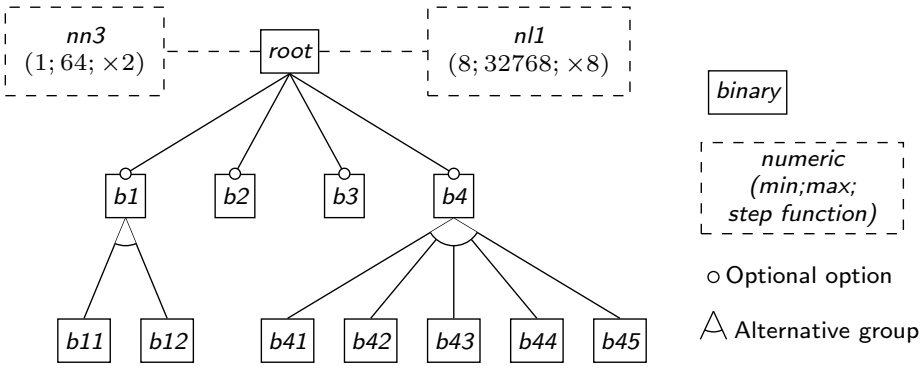


Figure A.14: Feature diagram for artificial DATASET_06

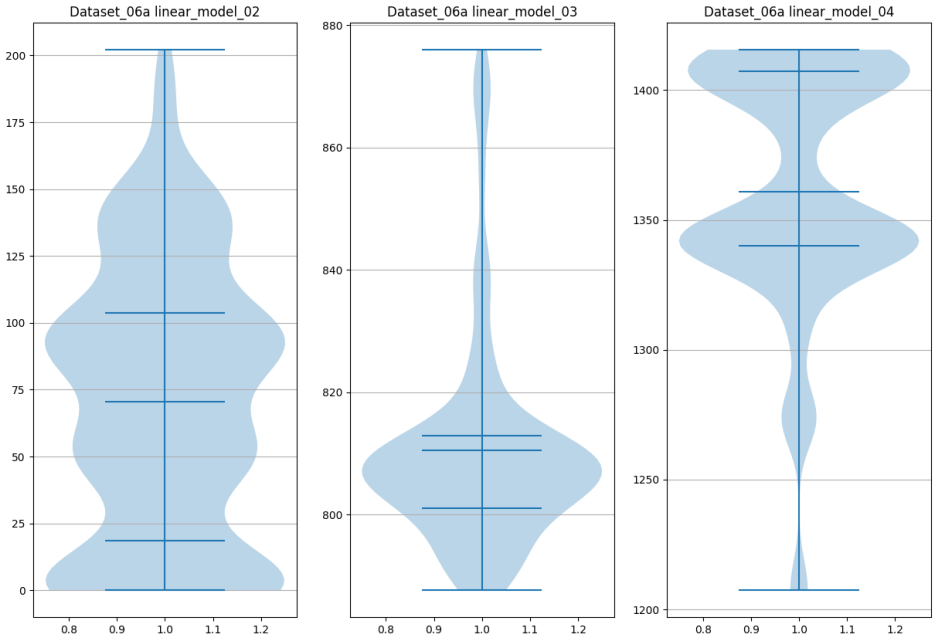


Figure A.15: Measurement distributions for artificial DATASET_06

Table A.1: Publications applying binary encoding mechanisms to deal with numeric options

No.	Title of publication	Authors	Year	Sampling strategy	Encoding	Determined by	Learning method	Contribution[1]
1	"Automated Search for Configurations of Deep Neural Network Architectures" [29]	Ghamizi et al.	2019	PLEDGE:Global Maximum Distance Prioritization	*One-hot*	Explicit		No
2	"Data-efficient Performance Learning for Configurable Systems" [37]	Guo et al.	2018	Random	*One-hot*	Case study HIPA^{CC}	CART	Yes
3	"Detecting Control-Flow and Performance Interactions in Highly-Configurable Systems" [20]	Denk	2017	Not mentioned	Omit feature values or use default	Explicit	MLR	Yes
4	"Distance-Based Sampling of Software Configuration Spaces" [47]	Kaltenecker et al.	2019	Random, solver-based, distance-based, tOW	*One-hot*	Case studies 7z, DUNE, HIPA^{CC}, JAVAGC, POLLY, VP9	MLR	Yes
5	"Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes" [30]	Gong and Chen	2022	Random	*One-hot*	Case studies MONGODB, LRZIP, TRIMESH, x264, EXASTENCILS	MLR, NN, DT, RF, kNN, KRR, SVR	No
6	"Evolution of Performance Influences in Configurable Systems" [40]	Hasreiter	2019	OW, PW, random	*One-hot*	Explicit	MLR	Yes
7	"Experiments on Optimizing the Performance of Stencil Codes with SPL Conqueror" [31]	Grebhahn et al.	2014	OW, PW, higher-order heuristic and hot-spot heuristic	*One-hot*	Explicit	MLR	Yes
8	"Grammar-Based Sampling" [90]	Weis	2020	Grammar-Based Sampling, tOW, solver-based, distance-based, random	*One-hot*	Case studies 7z, DUNE, HIPA^{CC}, JAVAGC, POLLY, VP9	MLR, RF, SVR	Yes

No.	Title of publication	Authors	Year	Sampling strategy	Encoding	Determined by	Learning method	Contr.
9	"Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems" [43]	Jamshidi et al.	2018	Active sampling (L2S)	Mapping of 2 values to 0;1	Explicit	MLR	No
10	"Mastering Uncertainty in Performance Estimations of Configurable Software Systems" [21]	Dorn, Apel, and Siegmund	2020	OW, PW, 3OW	*One-hot*	Case studies HIPA^{CC}, HSQLDB, JAVAGC, POLLY, POSTGRESQL, VP9	Bayesian model	Yes
11	"Optimizing Performance of Stencil Code with SPL Conqueror" [34]	Grebhahn et al.	2014	OW, PW, higher-order heuristic and hot-spot heuristic	*One-hot*	Case studies HIPA^{CC}, HSMGP	MLR	Yes
12	"Performance Modeling under Resource Constraints Using Deep Transfer Learning" [57]	Marathe et al.	2017	random	*One-hot*	Explicit	Extra Trees, RF, DFNN for transfer	No
13	*Scalable Performance Models for Highly Configurable Systems* [71]	Oh and Zilmano	2020	Random	*One-hot*	Case studies DUNE, HIPA^{CC}, HSMGP	DFNN	No
14	"Tradeoffs in modeling performance of highly configurable software systems" [49]	Kolesnikov et al.	2019	Not mentioned	*One-hot*	Case studies DUNE, HSMGP	MLR	Yes
15	"White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems" [88]	Velez et al.	2021	Pick > 1 $\mathbf{c}$ per subspace	Mapping of 2 values to 0;1	Explicit	Composition of local linear models	Yes

1 This column indicates whether this university Chair on its various locations has participated in composing this publication.

Table A.2: Publications processing numeric options directly

No.	Title of publication	Authors	Year	Sampling strategy		Learning method	Contribution
				Binary	Numeric		
1	"ACTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks" [7]	Bao et al.	2019	Random		GAN	No
2	"An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems" [42]	Jamshidi and Casale	2016	Bayesian Optimization with Latin Hypercube Design and GPR			No
3	"Automatic Configuration of the Cassandra Database using irace" [81]	Silva-Muñoz, Franzin, and Hugues	2021	irace:random iterative sampling		irace:stochastic optimizer	No
4	"Black-Box Models for Non-Functional Properties of AI Software Systems" [27]	Friesel and Spinczyk	2022	Not mentioned		DECART, CART, LMT, XGBoost, DECARTnb, LMTnb	No
5	"Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines" [41]	Henard et al.	2015	Solver-based		Multi-Objective Optimization	No
6	"Comparison of Analytical and Empirical Performance Models: A Case Study on Multigrid Systems" [45]	Kaltenecker	2016	Not mentioned	FF, PBD, CCID, random	MLR	Yes
7	"DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network" [38]	Ha and Zhang	2019	Random		DFNN	No
8	"Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes"[2] [30]	Gong and Chen	2022	Random		MLR, NN, DT, RF, kNN, KRR, SVR	No
9	"Experiments on Optimizing the Performance of Stencil Codes with SPL Conqueror"[2] [31]	Grebhahn et al.	2014	Function-learning heuristic		MLR	Yes
10	"Fast Performance Modeling across Different Database Versions Using Partitioned Co-Kriging" [13]	Cao et al.	2021	Random		kMeans, co-kriging	No
11	"Finding Faster Configurations Using FLASH" [65]	Nair et al.	2020	Bayesian Optimization with progressive, projective and rank-based sampling (random) and GPR			Yes

No.	Title of publication	Authors	Year	Sampling strategy		Learning method	Contribution
				Binary	Numeric		
12	"Hdconfigor: Automatically Tuning High Dimensional Configuration Parameters for Log Search Engines" [22]	Dou, Chen, and Zheng	2020	Bayesian Optimization with random-based sampling via Expected Improvement and GPR			No
13	"HINNPerf: Hierarchical Interaction Neural Network for Performance Prediction of Configurable Systems" [16]	Cheng, Gao, and Zheng	2022	Random		DFNN	No
14	"LONViZ: Unboxing the black-box of Configurable Software Systems from a Complex Networks Perspective" [52]	Li, Mao, and Chen	2022	ParamILS: random iterative approach		local optima network	No
15	"Multi-Objectivizing Software Configuration Tuning" [14]	Chen and Li	2021	Random-based objectivization		Plain Multi-objectivization Model, meta multi-objectivization	No
16	"Optimizing Performance of Stencil Code with SPL Conqueror" [2] [34]	Grebhahn et al.	2014	Function-learning heuristic		MLR	Yes
17	"Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning" [78]	Shu et al.	2020	Random		GAN	No
18	"Performance is not Boolean: Supporting Scalar Configuration Variables in NFP Models" [28]	Friesel and Spinczyk	2022	Random sampling with neighborhood exploration		CART, DECART, LMT, and Regression Model Tree	No
19	"Performance Prediction of Multigrid-Solver Configurations" [35]	Grebhahn et al.	2016	OW, PW	PBD, random	MLR	Yes
20	"Performance-Influence Models for Highly Configurable Systems" [79]	Siegmund et al.	2015	OW, NOW, PW	BBD, CCID, PBD, random	MLR	Yes
21	"Performance-Influence Models of Multigrid Methods: A Case Study on Triangular Grids" [32]	Grebhahn et al.	2017	OW, NOW, PW	CCID, PBD, DOD	MLR	Yes
22	"Predicting Performance of Software Configurations: There is no Silver Bullet" [33]	Grebhahn, Siegmund, and Apel	2019	OW, PW, 3OW, random	OFAT, BBD, CCID, PBD, DOD, random	CART, kNN, KRR, MLR, RF, SVR	Yes
23	"Regression Models for Performance Ranking of Configurable Systems: A Comparative Study" [15]	Chen et al.	2020	Rank-based iterative (random)		CART, SVR,GPR, Gradient Boosted Regression Trees	No

No.	Title of publication	Authors	Year	Sampling strategy		Learning method	Contribution
				Binary	Numeric		
24	"The Interplay of Compile-Time and Run-Time Options for Performance Prediction" [51]	Lesoil et al.	2021	Random		RF	No
25	"The Interplay of Sampling and Machine Learning for Software Performance Prediction" [46]	Kaltenecker et al.	2020	OW, NOW, tOW, random	OFAT, BBD, CCID, PBD, DOD, random	CART, kNN, KRR, MLR, RF, SVR	Yes
26	"Using Bad Learners to Find Good Configurations" [64]	Nair et al.	2017	Progressive and projective sampling, rank-based iterative (random)		CART	Yes
27	"VEER: A Fast and Disagreement-Free Multi-objective Configuration Optimizer" [72]	Peng et al.	2021	Bayesian Optimization with random-based sampling approach and CART			Yes
28	"White-Box Performance-Influence Models: A Profiling and Learning Approach" [89]	Weber, Apel, and Siegmund	2021	OW, PW	PBD	CART	Yes

2 This article is found also in Table A.1, as it uses both binary encoding and numeric features for performance prediction.

Table A.3: Details of the selected real software systems as well as the generated artificial ones

No.	Software system	Case study name	Binary features		Numeric features[3]		$\lvert\mathbf{C}_{valid}\rvert$	NFP
1	7z	7z	10	filterOff; HeaderCompressionOff; mtOff; CompressionMethod; LZMA; LZMA2; PPMd; BZip2; Deflate; tmOff	3	Files(0; 100; +10); BlockSize(1; 4096; ×2); x(0; 10; +2)	68 640	Time, Size
2	BROTLI	BROTLI	0	-	2	WindowSize(10; 24; +1); CompressionLevel(0; 11; +1)	180	testmem, performance, decompressperf, size, testperf, memory, decompressmem, energy
3	DUNE	DUNE	8	Precon; Solver; SeqGS; SeqSOR; CGSolver; BiCGSTABSolver; LoopSolver; GradientSolver	3	pre(0; 6; +1); post(0; 6; +1); cells(50; 55; +1)	2 304	Performance
4	HSMGP	HSMGP	11	CGS; Smoother; CGS_IP_CG; CGS_IP_AMG; CGS_RED_AMG; Smoother_JAC; Smoother_GS; Smoother_GSAC; Smoother_GSACBE; Smoother_GSRB; Smoother_GSRBAC	3	pre(0; 6; +1); post(0; 6; +1); numCore(64; 4096; ×4)	3 456	AverageTimePerIteration, NumIterations, TimeToSolution
5	JAVAGC	JAVAGC_SMALL	5	DisableExplicitGC; UseAdaptiveGCBoundary; UseAdaptiveSizeDecayMajorGCCost; UseAdaptiveSizePolicy; UseAdaptiveSizePolicyFootprintGoal	6	MaxTenuringThreshold(5; 15; +5); MinSurvivorRatio(1; 10; +3); NewRatio(1; 32; ×2); SurvivorRatio(1; 31; +5); TenuredGenerationSizeSupplement(50; 90; +20); TenuredGenerationSizeSupplementDecay(2; 16; ×2)	193 536	GCTime

No.	Software system	Case study name	Binary features		Numeric features		$\lvert\mathbf{C}_{valid}\rvert$	NFP
6	Libopus	Libopus pervolution	15	mode; audio; cbr; cvbr; bandwidth; NB; WB; FB; frameSize; lowest; default; highest; forcemono; MB; SWB	4	channels(1; 2; +1); samplingRate(8000; 24000; +8000); bitRate(6000; 216000; ×6); complexity(0; 10; +5)	6 480	performance
7	lrzip	lrzip pervolution	6	compression; compressionBzip2; compressionGzip; compressionLzo; compressionLzma; disableCompressibilityTesting	3	level(1; 9; +1); maxWindowSize(1; 8; ×2); processorCount(1; 8; ×2)	1 440	performance, size, cpu
8	MySQL	MySQL MariaDB pervolution	8	delayedInnodbLogFlush; delayedInnodbLogWrite; innodbFlushMethod; fsyncFlush; dsyncFlush; directFlush; binaryLog; innodbDoubleWrite	3	tempTableSize(256; 65536; ×16); innodbBufferPoolSize(8; 512; ×8); innodbLogBufferSize(1; 64; ×8)	972	performance, cpu
9	OpenVPN	OpenVPN	18	tcp; compression; none; lzo; cipher; AES_256_CBC; AES_128_CBC; TCP_NODEAL; prng; prng_none; prng_rsa_sha512; prng_sha512; prng_sha1; auth; auth_none; auth_sha1; auth_sha512; auth_rsa_sha512	1	reneg_bytes(10; 70; +20)	512	throughput_server
10	Polly	Polly	15	polly_vectorizer; none; polly; polly_parallel; parallel; parallel_force; polly_no_tiling; polly_delinearize; polly_dependences_analysis_type; value_based; memory_based; polly_opt_fusion; min; max; polly_opt_simplify_deps	4	pollyrtcmaxparameters(1; 16; ×2); pollydefaulttilesize(4; 1024; ×4); pollyoptmaxconstantterm(1; 10000; ×10); pollyoptmaxcoefficient(1; 10000; ×10)	60 000	UserTime, ElapsedTime
11	PostgreSQL	PostgreSQL pervolution	5	fsync; synchronousCommit; fullPageWrites; trackActivities; trackCounts	3	sharedBuffers(64; 256; ×2); tempBuffers(2; 32; ×4); workMem(256; 4096; ×4)	864	performance, cpu

No.	Software system	Case study name	Binary features		Numeric features		$\lvert\mathbf{C}_{valid}\rvert$	NFP
12	PostgreSQL	PostgreSQL pervolution energy	5	fsync; synchronousCommit; fullPageWrites; trackActivities; trackCounts	3	sharedBuffers(64;256;×2); tempBuffers(2;32;×4); workMem(256;4096;×4)	864	performance, cpu, benchmark-energy, fixed-energy
13	VP8	VP8 pervolution energy	9	twoPass; quality; bestQuality; goodQuality; rtQuality; constantBitrate; autoAltRef; noAltRef; allowResize	4	threads(1;4;+1); tokenParts(0;2;+1); arnrMaxFrames{0;5;15}; arnrStrength(0;6;+3)	2 736	performance, size, energy, cpu
14	VP9	VP9 pervolution	11	twoPass; quality; bestQuality; goodQuality; rtQuality; constantBitrate; autoAltRef; noAltRef; allowResize; columnTiling; rowTiling	3	threads(1;8;×2); arnrMaxFrames{0;5;15}; arnrStrength(0;6;+3)	3 008	performance, size, cpu
15	x264	x264 pervolution	6	no_8x8dct; no_cabac; no_deblock; no_fast_pskip; no_mbtree; no_mixed_refs	3	rc_lookahead(10;160;×4); ref(1;16;×2); cores(1;8;×2)	3 840	performance, size, cpu
16	x265	x265 pervolution	17	asm; wpp; rdoLevel; rdoBasic; rdoSplits; rdoChroma; rdoPrediction; rect; amp; bitdepth; 8bit; 10bit; me; meDia; meHex; meUmh; meStar	2	frameThreads(1;8;×2); ref{1;3;6}	3 840	performance, size, cpu
17	-	Dataset_01	3	b1; b2; b3	3	nn2(6;48;+6); nk3(50;3200;×8); nn3(0;102;+17)	1 344	linear_model_[06,07,09,11]
18	-	Dataset_02	0	-	3	nl1(1;99;+2); nl2(300;1800;+300); nl4(5;115;+5)	6 900	linear_model_[02,04,07,10]
19	-	Dataset_03	5	b1; b2; b3; b4; b5	1	nl3(1;11;+1)	352	linear_model_[02,04,05,06]
20	-	Dataset_04	7	b1; b11; b2; b3; b4; b41; b42	1	nk1{2;27;54}	72	linear_model_[01,04,06]
21	-	Dataset_05	7	b1; b2; b3; b4; b41; b42; b43	1	nn1{0;8;16;32;48}	80	linear_model_[02,03,05]
22	-	Dataset_06	11	b1; b11; b12; b2; b3; b4; b41; b42; b43; b44; b45	2	nl1(8;32768;×8);nn3(1;64;×2)	1 400	linear_model_[02-04]

3 Regarding numeric features, their values are given in the following form (minimal value;maximal value;step function). In case this is not possible, their are given in set notation.

BIBLIOGRAPHY

[1] Mathieu Acher, Hugo Martin, Luc Lesoil, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Khelladi, Olivier Barais, and Juliana Pereira. "Feature Subset Selection for Learning Huge Configuration Spaces: The Case of Linux Kernel Size." In: *Proceedings of the International Systems and Software Product Line Conference (SPLC) - Volume A*. 2022, pp. 85–96.

[2] Bouzid Ait-Amir, Abdelkhalak El Hami, and Philippe Pougnet. "6 - Meta-Model Development." In: *Embedded Mechatronic Systems 2*. Ed. by Abdelkhalak El Hami and Philippe Pougnet. Elsevier, 2015, pp. 151–179.

[3] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. "IncLing: Efficient Product-Line Testing Using Incremental Pairwise Sampling." In: *Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE)* 52.3 (2016), pp. 144–155.

[4] Mokhtar Alaya, Simon Bussy, Stéphane Gaïffas, and Agathe Guilloux. "Binarsity: a penalization for one-hot encoded features in linear supervised learning." In: *Journal of Machine Learning Research* 20.118 (2019), pp. 1–34.

[5] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. 2013.

[6] Sven Apel, Sergiy S. Kolesnikov, Norbert Siegmund, Christian Kästner, and Brady Garvin. "Exploring feature interactions in the wild: the new feature-interaction challenge." In: *International Workshop on Feature-Oriented Software Development (FOSD)*. 2013, pp. 1–8.

[7] Liang Bao, Xin Liu, Fangzheng Wang, and Baoyin Fang. "ACTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks." In: *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 2019, pp. 465–476.

[8] Thorsten Berger. "Variability Modeling in the Real. An Empirical Journey from Software Product Lines to Software Ecosystems." PhD thesis. Germany: University of Leipzig, 2012.

[9] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[10] James Bergstra, Daniel Yamins, and David Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2013, pp. 115–123.

[11] George Box and Donald Behnken. "Simplex-Sum Designs: A Class of Second Order Rotatable Designs Derivable From Those of First Order." In: *The Annals of Mathematical Statistics* 31 (1960), pp. 838–864.

[12] George Box and K. Wilson. "On the Experimental Attainment of Optimum Conditions." In: *Breakthroughs in Statistics. Perspectives in Statistics*. Vol. 2. 1992, pp. 270–310.

[13] Rong Cao, Liang Bao, Shouxin Wei, Jiarui Duan, Xi Wu, Yeye Du, and Ren Sun. "Fast Performance Modeling across Different Database Versions Using Partitioned Co-Kriging." In: *Applied Sciences* 11.20 (2021).

[14] Tao Chen and Miqing Li. "Multi-Objectivizing Software Configuration Tuning." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2021, pp. 453–465.

[15] Yuntianyi Chen, Yongfeng Gu, Lulu He, and J. Xuan. "Regression Models for Performance Ranking of Configurable Systems: A Comparative Study." In: *Structured Object-Oriented Formal Language and Method (SOFL+MSVL)*. 2020.

[16] Jiezhu Cheng, Cuiyun Gao, and Zibin Zheng. "HINNPerf: Hierarchical Interaction Neural Network for Performance Prediction of Configurable Systems." In: *ACM Transactions on Software Engineering and Methodology* (2022).

[17] *Control variable. APA Dictionary of Psychology*. Oct. 6, 2022. URL: https://dictionary.apa.org/control-variable.

[18] Uwe Czienskowski. *Wissenschaftliche Experimente: Planung, Auswertung, Interpretation*. Beltz, Psychologie Verlag-Union, 1996.

[19] Anton Deitmar. *Analysis*. Springer, 2021.

[20] Alexander Denk. "Detecting Control-Flow and Performance Interactions in Highly-Configurable Systems." Master's Thesis. Germany: University of Passau, 2017.

[21] Johannes Dorn, Sven Apel, and Norbert Siegmund. "Mastering Uncertainty in Performance Estimations of Configurable Software Systems." In: *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 2020, pp. 684–696.

[22] Hui Dou, Pengfei Chen, and Zibin Zheng. "Hdconfigor: Automatically Tuning High Dimensional Configuration Parameters for Log Search Engines." In: *IEEE Access* 8 (2020), pp. 80638–80653.

[23] Clemens Dubslaff, Kallistos Weis, Christel Baier, and Sven Apel. "Causality in Configurable Software Systems." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2022, pp. 325–337.

[24] Computing Research & Education, ed. *CORE Conference Portal*. Jan. 19, 2022. URL: http://portal.core.edu.au/conf-ranks/?search=Software+Engineering&by=all&source=CORE2021&sort=arank&page=1.

[25] Bastian Fleischmann. "Meta-Learning for Performance Prediction on Configurable Software Systems." Bachelor's Thesis. Germany: University of Passau, 2018.

[26] Giles Foody, Ajay Mathur, Carolina Sanchez-Hernandez, and Doreen Boyd. "Training set size requirements for the classification of a specific class." In: *Remote Sensing of Environment* 104.1 (2006), pp. 1–14.

[27] Daniel Friesel and Olaf Spinczyk. "Black-Box Models for Non-Functional Properties of AI Software Systems." In: *International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 2022, pp. 170–180.

[28] Daniel Friesel and Olaf Spinczyk. "Performance is not Boolean: Supporting Scalar Configuration Variables in NFP Models." In: *Tagungsband des FG-BS Frühjahrstreffens 2022*. 2022.

[29] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. "Automated Search for Configurations of Deep Neural Network Architectures." In: *ArXiv* (2019).

[30] Jingzhi Gong and Tao Chen. "Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes." In: *International Conference on Mining Software Repositories (MSR)*. 2022, pp. 482–494.

[31] Alexander Grebhahn, Sebastian Kuckuk, Christian Schmitt, Harald Köstler, Norbert Siegmund, Sven Apel, Frank Hannig, and Jürgen Teich. "Experiments on Optimizing the Performance of Stencil Codes with SPL Conqueror." In: *Parallel Processing Letters* 24.3 (2014).

[32] Alexander Grebhahn, Carmen Rodrigo, Norbert Siegmund, Francisco Gaspar, and Sven Apel. "Performance-Influence Models of Multigrid Methods: A Case Study on Triangular Grids." In: *Concurrency and Computation: Practice and Experience* 29.17 (2017).

[33] Alexander Grebhahn, Norbert Siegmund, and Sven Apel. "Predicting Performance of Software Configurations: There is no Silver Bullet." In: *ArXiv* (2019).

[34] Alexander Grebhahn, Norbert Siegmund, Sven Apel, Sebastian Kuckuk, Christian Schmitt, and Harald Köstler. "Optimizing Performance of Stencil Code with SPL Conqueror." In: *Proceedings of the International Workshop on High-Performance Stencil Computations (HiStencils)*. 2014, pp. 7–14.

[35] Alexander Grebhahn, Norbert Siegmund, Harald Köstler, and Sven Apel. "Performance Prediction of Multigrid-Solver Configurations. Lecture Notes in Computational Science and Engineering." In: *Software for Exascale Computing (SPPEXA)*. 2016, pp. 69–88.

[36] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. "Variability-aware performance prediction: A statistical learning approach." In: *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 2013, pp. 301–311.

[37] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wąsowski, and Huiqun Yu. "Data-efficient Performance Learning for Configurable Systems." In: *Empirical Software Engineering (EMSE)* 23.3 (2018), pp. 1826–1867.

[38] Huong Ha and Hongyu Zhang. "DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2019, pp. 1095–1106.

[39] Huong Ha and Hongyu Zhang. "Performance-Influence Model for Highly Configurable Software with Fourier Learning and Lasso Regression." In: *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019, pp. 470–480.

[40] Johannes Hasreiter. "Evolution of Performance Influences in Configurable Systems." Master's Thesis. Germany: University of Passau, 2019.

[41] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. "Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2015, pp. 517–528.

[42] Pooyan Jamshidi and Giuliano Casale. "An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems." In: *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2016, pp. 39–48.

[43] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. "Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2018, pp. 71–82.

[44] Enio Jelihovschi, José Faria, and Ivan Allaman. "ScottKnott: A Package for Performing the Scott-Knott Clustering Algorithm in R." In: *Trends in Computational and Applied Mathematics* 15.1 (Mar. 2014), pp. 3–17.

[45] Christian Kaltenecker. "Comparison of Analytical and Empirical Performance Models: A Case Study on Multigrid Systems." Master's Thesis. Germany: University of Passau, 2016.

[46] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. "The Interplay of Sampling and Machine Learning for Software Performance Prediction." In: *IEEE Software* 37.4 (2020), pp. 58–66.

[47] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. "Distance-Based Sampling of Software Configuration Spaces." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2019, pp. 1084–1094.

[48] Geoffrey Keppel and Thomas Wickens. *Design and analysis: a researchers handbook*. 4th ed. Pearson Prentice Hall, 2004.

[49] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. "Tradeoffs in modeling performance of highly configurable software systems." In: *Software & Systems Modeling* 18.3 (2019), pp. 2265–2283.

[50] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. "YASA: Yet Another Sampling Algorithm." In: *Proceedings of the International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS)*. 2020, pp. 1–10.

[51] Luc Lesoil, Mathieu Acher, Xhevahire Térnava, Arnaud Blouin, and Jean-Marc Jézéquel. "The Interplay of Compile-Time and Run-Time Options for Performance Prediction." In: *Proceedings of the International Systems and Software Product Line Conference (SPLC) - Volume A*. 2021.

[52] Ke Li, Peili Mao, and Tao Chen. "LONViZ: Unboxing the black-box of Configurable Software Systems from a Complex Networks Perspective." In: *ArXiv* (2022).

[53] Jörg Liebig. "Analysis and Transformation of Configurable Systems." PhD thesis. Germany: University of Passau, 2015.

[54] Scott Lundberg, Gabriel Erion, and Su-In Lee. "Consistent Individualized Feature Attribution for Tree Ensembles." In: *ArXiv* (2018).

[55] Scott Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions." In: *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 4765–4774.

[56] H. Mann and D. Whitney. "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other." In: *The Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60.

[57] Aniruddha Marathe, Rushil Anirudh, Nikhil Jain, Abhinav Bhatele, Jayaraman Thiagarajan, Bhavya Kailkhura, Jae-Seung Yeom, Barry Rountree, and Todd Gamblin. "Performance Modeling under Resource Constraints Using Deep Transfer Learning." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2017.

[58] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. "A Comparison of 10 Sampling Algorithms for Configurable Systems." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2. 2016, pp. 643–654.

[59] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2022. Chap. 8 Local Model-Agnostic Methods, pp. 179–240.

[60] Stefan Mühlbauer, Sven Apel, and Norbert Siegmund. "Identifying Software Performance Changes across Variants and Versions." In: *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 2020, pp. 611–622.

[61] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. "Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features." In: *Proceedings of the International Systems and Software Product Line Conference (SPLC) - Volume A*. 2019, pp. 289–301.

[62] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. "Where Do Configuration Constraints Stem From? An Extraction Approach and an Empirical Study." In: *IEEE Transactions on Software Engineering* 41.8 (2015), pp. 820–841.

[63] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. "Faster Discovery of Faster System Configurations with Spectral Learning." In: *ArXiv* (2017).

[64] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. "Using Bad Learners to Find Good Configurations." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2017, pp. 257–267.

[65] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. "Finding Faster Configurations Using FLASH." In: *IEEE Transactions on Software Engineering* 46.7 (2020), pp. 794–811.

[66] Damir Nešić, Jacob Krüger, Stefan Stănciulescu, and Thorsten Berger. "Principles of Feature Modeling." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2019, pp. 62–73.

[67] Son Nguyen, Hoan Nguyen, Ngoc Tran, Hieu Tran, and Tien N. Nguyen. "Feature-Interaction Aware Configuration Prioritization for Configurable Code." In: *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 2019, pp. 489–501.

[68] Thanh Nguyen, Ugur Koc, Javran Cheng, Jeffrey Foster, and Adam Porter. "IGen: Dynamic Interaction Inference for Configurable Software." In: *Proceedings of the Symposium on the Foundations of Software Engineering (FSE)*. 2016, pp. 655–665.

[69] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund. "Finding near-optimal configurations in product lines by random sampling." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2017, pp. 61–71.

[70] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. "Finding Near-Optimal Configurations in Product Lines by Random Sampling." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2017, pp. 61–71.

[71] Jeho Oh and Oleg Zilmano. *Scalable Performance Models for Highly Configurable Systems*. 2020.

[72] Kewen Peng, Christian Kaltenecker, Norbert Siegmund, Sven Apel, and Tim Menzies. "VEER: A Fast and Disagreement-Free Multi-objective Configuration Optimizer." In: *ArXiv* (2021).

[73] R. Plackett and J. Burman. "The Design of Optimum Multifactorial Experiments." In: *Biometrika* 33 (4 1946), pp. 305–325.

[74] Philipp Probst, Marvin Wright, and Anne-Laure Boulesteix. "Hyperparameters and tuning strategies for random forest." In: *WIREs Data Mining and Knowledge Discovery* 9.3 (2019).

[75] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction Design - Beyond Human-Computer Interaction*. Wiley, 2019.

[76] A. J. Scott and M. Knott. "A Cluster Analysis Method for Grouping Means in the Analysis of Variance." In: *Biometrics* 30.3 (1974), pp. 507–512.

[77] Lloyd Shapley. "17. A Value for n-Person Games." In: *Contributions to the Theory of Games*. Vol. II. Princeton University Press, 1953, pp. 307–317.

[78] Yangyang Shu, Yulei Sui, Hongyu Zhang, and Guandong Xu. "Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning." In: *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2020, pp. 1–11.

[79] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. "Performance-Influence Models for Highly Configurable Systems." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2015, pp. 284–294.

[80] Norbert Siegmund, Stefan Sobernig, and Sven Apel. "Attributed Variability Models: Outside the Comfort Zone." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2017, pp. 268–278.

[81] Moisés Silva-Muñoz, Alberto Franzin, and Bersini Hugues. "Automatic Configuration of the Cassandra Database using irace." In: *PeerJ Computer Science* 7.e634 (2021).

[82] Werner Stangl. *Kontrollvariablen. Online Lexikon für Psychologie und Pädagogik*. Oct. 6, 2022. URL: https://lexikon.stangl.eu/5504/kontrollvariablen.

[83] Werner Stangl. *Verkettung. Mathematisches Lexikon*. Oct. 22, 2022. URL: https://www.mathe-online.at/mathint/lexikon/v.html#Verkettung.

[84] M. Sujitha and N. Sivakumar. "Software Effort Estimation Using Scott Knott Test." In: *International Journal of Computer Science and Engineering Communications* 3 (2 2015).

[85] Chico Sundermann, Thomas Thüm, and Ina Schaefer. "Evaluating SAT Solvers on Industrial Feature Models." In: *Proceedings of the International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS)*. 2020.

[86] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. "A Classification and Survey of Analysis Strategies for Software Product Lines." In: *ACM Computing Surveys* 47.1 (2014).

[87] Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, and Christian Kästner. "ConfigCrusher: towards white-box performance analysis for configurable systems." In: *Automated Software Engineering*. Vol. 27. 3. 2020, pp. 265–300.

[88] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. "White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2021, pp. 1072–1084.

[89] Max Weber, Sven Apel, and Norbert Siegmund. "White-Box Performance-Influence Models: A Profiling and Learning Approach." In: *Proceedings of the International Conference on Software Engineering (ICSE)*. 2021, pp. 1059–1071.

[90] Kallistos Weis. "Grammar-Based Sampling." Master's Thesis. Germany: Saarland University, 2020.

[91] Frank Wilcoxon. "Individual Comparisons by Ranking Methods." In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83.

[92] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. "Hey, You Have given Me Too Many Knobs!: Understanding and Dealing with over-Designed Configuration in System Software." In: *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 2015, pp. 307–319.

[93] H. Ye and H. Liu. "Approach to modelling feature variability and dependencies in software product lines." In: *IEE Proceedings - Software* 152 (3 2005), pp. 101–109.

[94] Zhihong Zhang and Bai Xiaofeng. "Comparison about the Three Central Composite Designs with Simulation." In: *Proceedings of the International Conference on Advanced Computer Control (ICACC)*. 2009, pp. 163–167.