Generating Attributed Variability Models for Transfer Learning

Johannes Dorn
Bauhaus-Universität Weimar
johannes.dorn@uni-weimar.de

Sven Apel
Saarland University
apel@cs.uni-saarland.de

Norbert Siegmund
Bauhaus-Universität Weimar
norbert.siegmund@uni-weimar.de

ABSTRACT

Modern software systems often provide configuration options for customizing of the system's functional and non-functional properties, such as response time and energy consumption. The valid configurations of a software system are commonly documented in a variability model. Supporting the optimization of a system's non-functional properties, variability models have been extended with attributes that represent the influence of one or multiple options on a property. The concrete values of attributes are typically determined only in a single environment (e.g., for a specific software version, a certain workload, and a specific hardware setup) and are applicable only for this context. Changing the environment, attribute values need to be updated. Instead of determining all attributes from scratch with new measurements, recent approaches rely on transfer learning to reduce the effort of obtaining new attribute values. However, the development and evaluation of new transfer-learning techniques requires extensive measurements by themselves, which often is prohibitively costly. To support research in this area, we propose an approach to synthesize realistic attributed variability models from a base model. This way, we can support research and validation of novel transfer-learning techniques for configurable software systems. We use a genetic algorithm to vary attribute values. Combined with a declarative objective function, we search a changed attributed variability model that keeps some key characteristics while mimicking realistic changes of individual attribute values. We demonstrate the applicability of our approach by replicating the evaluation of an existing transfer-learning technique.

CCS CONCEPTS

• **Software and its engineering** → *Software performance*; • **Computing methodologies** → *Genetic algorithms*; *Transfer learning*;

KEYWORDS

Variability modelling, attributed variability models, transfer learning, Loki

ACM Reference Format:
Johannes Dorn, Sven Apel, and Norbert Siegmund. 2020. Generating Attributed Variability Models for Transfer Learning. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS '20), February 5–7, 2020, Magdeburg, Germany.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3377024.3377040

1 INTRODUCTION

Configurable software systems can be customized to meet different use cases by setting binary and numeric configuration options. Usually, a *variability model* (e.g., a feature model [1]) describes all configuration options and constraints among them. A *configuration* specifies a system variant's functional and non-functional properties. This way, users can tailor the software system not only according to their functional requirements, but also optimize it with respect to non-functional properties, such as response time, memory size, and energy consumption.

Optimizing non-functional properties is a non-trivial task and has been subject to considerable research [7, 8, 15, 16, 19, 24, 26, 27]. Most approaches rely on a surrogate model in the form of an *attributed variability model (AVM)*, which specifies not only the variability of the configurable system, but also assigns attributes to individual options and interactions among options. Figure 1 depicts an exemplary AVM with energy consumption as attribute. Using all individual options' attribute values, we can compute the resulting configuration's value by summing up the attribute values of selected options. Equation 1 formalizes this computation as a polynomial of options $o_1, \ldots, o_n \in \{0, 1\}$ that form a configuration C_i from the set of all valid configurations C. The coefficient α_i represent the influence of o_i on the attribute and α_0 represents the base values of the software system. Function $\pi : C \to \mathbb{R}$ maps a given configuration to a real value quantifying the property in question:

$$\pi(C_i) = \alpha_0 + \alpha_1 \cdot o_1 + \alpha_2 \cdot o_2 + \cdots + \alpha_n \cdot o_n \tag{1}$$

Note that additional terms for interactions among options might be added as software systems usually have non-linear non-functional behavior.

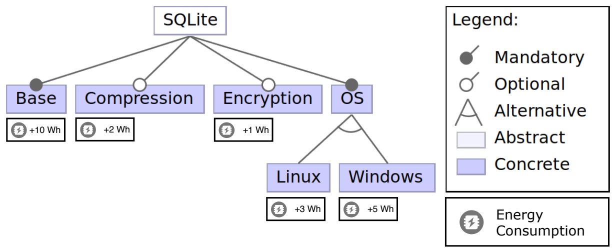

Figure 1: Simplified attributed variability model of the database system SQLite. Each option shows the estimated energy consumption in terms of its attribute value.

AVMs support optimization as they evaluate attribute values of individual configurations without expensive measurement of the corresponding variant. Beside optimization, developers may use AVMs to analyze which options slow down their system and which variant fits on memory-restricted devices.

Typically, an AVM represents attribute values of a single environment [6, 10–12]. That is, the values have been determined by measuring the software system in a specific version, with a specific workload, and on specific hardware. As soon as the environment changes, one cannot guarantee the soundness of the AVM anymore since the attribute values may not reflect the specifics of the new environment. Unfortunately, software is in constant change, and different customers have different workloads and hardware settings, rendering a static AVM insufficient.

Current practice is to obtain an entirely new AVM for the changed environment by measuring non-functional properties again for the same variants in the new environment and afterwards deducing the changed attribute values, for instance, via machine learning [5, 17, 20]. Clearly, this does not scale for frequent environmental changes and is also not efficient as always some information remains relevant in the new environment. For example, Valov et al. have shown that a change in hardware often only shifts the options' influences by a linear amount [25]. Their relative importance and ranking stay the same. Jamshidi et al. observed in an empirical study that performance distributions, relative importance of options and interactions, and the ranking of configurations with respect to performance maintains some characteristics for various environmental changes of configurable software systems [10].

Transfer learning is a technique for transferring knowledge from a source model to a target model such that only few additional measurements need to be conducted in the changed (target) environment. The soundness of the new model depends on the quality of the transfer-learning technique and its applicability to the specific case. Hence, transfer-learning techniques must be evaluated carefully to find the best target model. Evaluating a transfer-learning technique can be done by assessing the accuracy of the target model for a representative set of environment changes. But here comes the caveat: To extensively evaluate the AVM's accuracy, we would need to measure all variants across all environments. Clearly, this is infeasible even for medium-sized configurable software systems. Here, we struggle not only with the complexity of a software system's configuration space, but also with an exponential number of environments affecting the system's non-functional properties. Thus, the experimental effort for evaluating transfer-learning techniques in the field of configurable software systems is even more challenging than for optimization techniques for single AVMs. Moreover, for the development of new techniques in a single environment, researchers often rely on synthetic, non-realistic attribute values due to the huge measurement effort [21]. This can lead to biased and non-optimal techniques when applied to realistic settings [21]. We expect that this problem even grows for the development of transfer-learning techniques.

To address these problems and ease the development and evaluation of new practical transfer-learning techniques, we propose Loki, a tool that synthesizes a realistic target AVM that is related to, but different from a given source AVM. Loki supports the evaluation of transfer-learning techniques on several synthetic AVMs and requires only a single source AVM, which can be based on measurements. Using user specifications regarding the target AVM, we build a set of candidate AVMs by modifying the source AVM with operators that exist for certain user specifications. Next, we use a genetic algorithm to find candidate AVMs with attribute values that best satisfy all user specifications, including those without applicable operators.

Suppose the AVM of the database system in Figure 1 has been built based on measurements on cheap cluster machines, but a company intents to deploy it on high-performance server machines. The company could generate a range of AVMs with different degrees of relatedness to the source AVM in Figure 1 and test their transfer-learning technique against the resulting pairs of source and target models. They can also apply optimization on the target AVMs to infer which configurations are suitable in range of environments. This way, a subsequent optimization of the database system in different environments can be simulated upfront.

Technically, we re-implemented the tool Thor[1] [21] as an optional pre-step to start with a realistic AVM in cases where actual measurements are not possible to conduct at all. To summarize, our contributions are as follows:

- an approach to synthesize a realistic target AVM that is related to a given source AVM to support the development and evaluation of transfer-learning techniques;
- Loki, an open-source[2] tool that implements our approach;
- a replication of the existing transfer-learning approach *Learn to Sample* [11] using Loki to demonstrate its applicability.

2 PRELIMINARIES AND RELATED WORK

2.1 Transfer Learning

Attribute values in an AVM are either assigned manually based on user experience or can be determined via machine learning from a set of measurements, referred to as a *training set*. Typically, the training set is acquired in a single fixed environment; consequently, the inferred AVM is expected to be valid for the respective environment only, and likely to perform worse for other environments. Among several components, an environment is composed of (1) the hardware setup that runs the software system, (2) the workload that is executed, and (3) the version of the software system.

In the worst case, for each environment, a new AVM has to be obtained from scratch with entirely new measurements. By contrast, transfer-learning techniques leverage knowledge from an existing model to infer a new model more data-efficiently.

Different approaches to transfer learning have been proposed in the past. Jamshidi et al. developed *Learning to Sample (L2S)*, an approach that identifies influential options for an existing AVM and guides measurements in the new environment to reuse information on these options [11]. Valov et al. use a small training set from the new environment to infer a linear transfer function for the existing AVM, resulting in an AVM for the new environment [25]. Among a guided-measurement, linear-transfer and tree-based non-linear transfer approach, Iqbal et al. found the guided-sampling approach

[1]Thor allows users to generate an AVM based on a given variability model and a specification of attribute values and interactions. The generated values mimic the ones found in real-world software systems.

[2]Download and contribute to Loki at https://github.com/digital-bauhaus/Loki

to be superior for transfer learning hyper parameters of deep neural networks across different machines [9].

Reasons for the effectiveness of transfer learning were found in an exploratory study [10] and confirmed by a causal analysis [12]:

- For non-severe hardware environment changes, a strong linear correlation has been found for option attribute values.
- Across all studied environment changes, the distribution of variant attribute values stays similar as measured with the Kullback-Leibler-Divergence [2].
- Rank correlation of the variant attribute values proved to be even more robust than Kullback-Leibler-Divergence.
- Half of all options and 6 – 28% of all possible interactions were classified as influential and remained influential across environments.
- The attribute values for the investigated interactions depicted a high linear correlation.

To the best of our knowledge, there exists no test-bed creation tool for transfer learning that fits into a researcher's experimental pipeline, despite active research in the field of configurable software systems. Such a tool should provide means to synthesize AVMs based on a given AVM and reflect the aspects listed above.

2.2 State-of-the-Art AVM Synthesis

Loki is neither the first tool to synthesize AVMs nor is it the first to target transfer learning. In the following, we describe existing and missing capabilities of publicly available tools for synthesizing AVMs.

Thor. Thor [21] is designed to support the evaluation of algorithms and tools operating on a single AVM. It enables users to synthesize attribute values for arbitrarily large and constrained target variability models while behaving like a supplied source AVM. Thor uses the genetic algorithm NSGA-II [4] to optimize distribution similarities of option and interaction attribute values as well as variant attribute values to synthesize a realistic AVM. Varying the target model's number of options and interactions, it is possible to test an algorithm's fitness on problems with different complexities.

Loki covers the functionality of Thor while adding support for transfer learning, which we describe in Section 3.

GenPerf. Jamshidi et al. use the tool GenPerf to evaluate their transfer-learning approach [11]. Given a source AVM, GenPerf synthesizes a target AVM that covers a subset of aspects observed with real pairs of source and target AVMs. GenPerf is able to synthesize target AVMs whose option attribute values correlate to a (user-specified degree) with the given source AVM and, at the same time, keeps the variant attribute distributions of both AVMs similar. In this process, GenPerf changes attribute values for options and interactions, while setting the attribute value for some attributes and interactions to zero and increasing the value of others significantly in a stochastic manner; thus, not all influential options stay influential across environments and previously non-influential options may become influential [11]. However, GenPerf does not offer the selective change of only the most influential options and interaction attribute values nor a way to ensure rank correlation between source and target variant attribute values. In addition, the user needs to define the number of desired influential options and interactions as program variables, which requires the user to have detailed knowledge of the source AVM and GenPerf's internals. This is, however, an unrealistic requirement.

BeTTy. While Thor generates attributions for variability models based on existing AVMs, BeTTy [18] lets the user specify most parameters for AVM generation. Attribute distributions may be specified for options and interactions as well as parameters for constraints and size of the target model. BeTTy does not provide a mechanism to ensure a realistic distribution of variant attribute values and does not support the generation of changed AVMs from original AVMs.

3 SYNTHESIZING TARGET AVMS WITH LOKI

3.1 Overview

With Loki, we aim at applying realistic changes to a source AVM, yielding a target model that may be used to evaluate transfer-learning approaches. To be realistic, we pursue the synthesis of all common aspects among environment changes, as laid out in Section 2.1. With Loki, users are able to create a highly controlled test bed in which they can control the degrees and types of changes they want to see reflected in the target model.

Figure 2 visualizes the steps of our approach. Before synthesizing a target model, the user may use the included Thor functionality to derive a large source AVM from a given smaller AVM or common variability model (step 0). Using the synthesized AVM or an AVM based on real measurements, Loki first forms an initial set of candidate AVMs following the process described in Section 3.2 (step 1). This set AVMs forms the initial population for the optimization step (step 2) using a genetic algorithm (GA), which we explain in Section 3.3. Steps 1 and 2 are responsible for satisfying user requirements regarding the conformity with realistic aspects of environment changes as well as user-defined challenges to evaluate transfer-learning techniques. Finally, Loki exports all AVMs of the final population and provides analytic plots as well as a python module that provides a sampling API for the new AVMs (step 3). Alternatively, the user can constrain the export to a single AVM that has the highest score according to a weighted average of all fitness functions.

User specifications and optional expert settings are stated in a single YAML configuration file, whose format is documented at Loki's online repository. In the following, we describe steps 1 and 2 of our approach in more detail. We refer to Siegmund et al. [21] for details on the Thor functionality in step 0.

3.2 Generating Candidate AVMs

In step 1, Loki obtains a set of initial candidate AVMs, which forms the starting point for the GA in step 2. The process starts by evaluating the user specifications for generating the AVM. The user can specify the following:

- Aspects for the target AVM (see below);
- Custom selection of options and interactions to change;
- Internal parameters for tailoring the optimization process.

Aspects describe relations between source and target AVM, in particular:

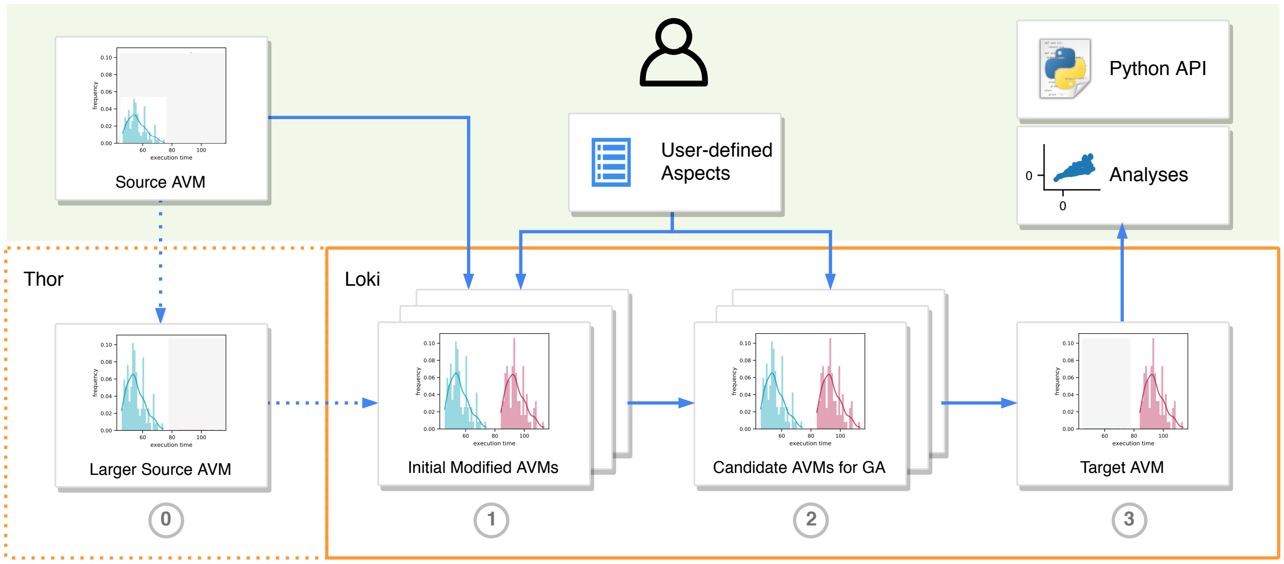

Figure 2: Workflow of Loki: (0) optional AVM extension with Loki's implementation of Thor, (1) directly apply aspects to build initial population for the genetic algorithm, (2) optimize all aspects with the genetic algorithm, (3) chose final AVM(s).

- The level of Gaussian noise to be applied;
- The linear transformation or
- Exponential transformation of attribute values to be applied;
- The similarity of option, interaction, and variant attribute values for source and target AVM given a desired similarity metric;
- The target degree of similarity ranging 0 – 1.

The user can chose from the following similarity metrics:

- Pearson's r [23]
- Spearman rank-order correlation [22]
- Euclidean Distance
- Kullback-Leibler-Divergence [14]
- Kolmogorov–Smirnov test statistic [3]

NSGA-II — the GA that we use — additionally has configurable parameters. Loki exposes these parameters for expert users; however, these parameters can stay default. The configurable NSGA-II parameters are the following:

- Population size
- Budget
- Cross-over strategy
- Number of reference variants used to assess fitness functions
- Sampling strategy for reference variants

We implemented genetic operators to change the source AVM in way that we meet all user specifications. Also, some specifications represent configuration options for the optimization process. Since some specifications might provide a tradeoff or have conflicting effects on the attribute values, we need to optimize for all objectives (i.e., specifications) using a weighted fitness function. By default, the GA applies the operators in the order they are listed. If another order is desired, Loki needs to be executed once per individual user specification. By repeatedly applying the genetic operators on the source AVM, we obtain a set of candidate AVMs. Next, we describe the genetic operators.

Formalization. We represent an AVM as a set of options and interactions: O where option $o \in O$. Without loss of generality, we model interactions similar to options. The only difference is that an interaction cannot be set by a user or sampling approach; instead, it is selected *iff* all options that the interaction corresponds to are selected. Furthermore, function $\alpha : O \to \mathbb{R}$ maps for a given configuration option or interaction to the corresponding attribute value represented as a real number. In other words, α returns for option o the coefficient α_o. We further define function $\Pi : \mathbb{P}(C) \to \mathbb{R}^{|C|}$ which enumerates the set of all configurations and produces a real-valued set containing the attribute values of all variants. Internally, we compute it as follows:

$$\Pi(C) = \{\pi(c) \mid c \in C\} \tag{2}$$

In general, each operator has a user-specified per-option and per-interaction probability p to be applied. This way, we realize a stochastic and iterative process to change an AVM, making the final model less deterministic and posing a challenge for the learning technique. Moreover, we can combine this probability with a filter such that the following specification is possible: Apply large noise (50%) randomly to 20% of the 50% most-influential options and interactions. Options and interactions with higher absolute coefficient value α_o are considered to be more influential with an AVM according to Equation 1.

Transformation Operator. Users may specify a linear or exponential transformation of attribute values from the source AVM to the target AVM. That is, we change the attribute values of options and interactions to meet the specified transformation. The factor λ is multiplied with each attribute value $\alpha(o)_s$ to yield $\alpha(o)_t$:

$$\forall o \in O' : \alpha(o)_t = \lambda \cdot \alpha(o)_s, \tag{3}$$

where $O' \subseteq O$. The user specifies whether all options' values should be transformed $O' = O$ or only the top $p\%$ most-influential options and interactions $O' \subset O$.

Recall that linear correlation is one of the aspects of a realistic environment change [10]. Since linear transformation causes linear correlation, this specification is useful to evaluate whether a transfer-learning approach harnesses this aspect. Conversely, the user can specify an exponential mapping. In this case, λ is used as exponent for $\alpha(o)_s$ to compute a new $\alpha(o)_t$:

$$\forall o \in O : \alpha(o)_t = \alpha(o)_s^{\lambda} \tag{4}$$

Since an exponential relationship has not been observed to be an aspect of realistic environment changes, this transformation is useful to challenge a transfer-learning technique.

Gaussian-Noise Operator. Jamshidi et al. observed different types of attribute-value correlations for source and target AVMs, although none were perfect correlations [10]. To mimic imperfect correlations, Loki's supports the application of noise on attribute values of the source model. The operator for Gaussian noise relies on the bell-shaped Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. It is centered around its mean μ, while its shape is defined by its standard deviation σ. Loki's Gaussian-noise operator is configured with a relative noise σ_{rel} and draws a new value for each attribute value according to Equation 5. Thus, larger values are likely to be changed by a larger margin.

$$\alpha(o)_t = \mathcal{N}(\alpha(o)_s, (\sigma_{rel} \cdot \alpha(o)_s)^2) \tag{5}$$

3.3 Optimizing Aspects

As described before, the user can specify different aspects for the new AVM to be met, such as transformations, similarities of distributions, noise, and so on. As these aspects might be conflicting, there is no single AVM that meets all specifications, but a range of AVMs that favor some aspects over others. Hence, we define the search for a target AVM as a multi-objective optimization problem containing one objective per specification. In the following, we abbreviate the source and target AVM with AVM_s and AVM_t, respectively and formalize the optimization using the following fitness function:

$$\begin{aligned} \max f(AVM_s, AVM_t) = \;& \beta_0 \cdot f_1(AVM_s, AVM_t) \\ +\;& \beta_1 \cdot f_2(AVM_s, AVM_t) \\ +\;& \dots \\ +\;& \beta_n \cdot f_n(AVM_s, AVM_t), \end{aligned} \tag{6}$$

where $f_1 \dots f_n$ represent individual fitness functions that assess the fitness of an individual aspect and $\overline{\beta}$ represents the user-defined weighting of the individual aspects.

We use the genetic algorithm NSGA-II [4] to solve this optimization problem. It starts with the initial population of step 1 and iteratively applies genetic operators to combine candidate AVMs, assesses the combined AVMs by means of our fitness functions, and finally selects the best candidates for the next iteration. We stop the GA after a given budget of iterations or after three iterations without improvement. To exemplify how we compute an individual fitness score, we explain the computations of the fitness function for Gaussian noise and the distribution similarity.

Gaussian-Noise Fitness. We assess how well the Gaussian-Noise specification is fulfilled by comparing the relative option and interaction attribute value differences between the source AVM and the respective candidate AVM against the Gaussian distribution specified by the user with Pearson's r [23]:

$$\Delta_\alpha = \{(\alpha_i^{target} - \alpha_i^{source})/\alpha_i^{source} \mid \alpha_i \in AVM_s\} \tag{7}$$

$$f_{noise} = \rho(\Delta_\alpha, \mathcal{N}) \tag{8}$$

Distribution-Similarity Fitness. Loki is the first tool in this field that allows for the specification and evaluation of multiple distribution distance metrics, such as (1) Pearson's correlation coefficient to ensure linear correlation [23], (2) Spearman's rank correlation coefficient for high rank correlation, (3) Euclidean Distance, (4) Kolmogorov–Smirnov test statistic, and (5) the Kullback-Leibler-Divergence to keep option interaction, and variant attribute values of the target model similar to the source model [11]. If the user chooses multiple distance metrics, their average is used as the fitness value.

We use as a fitness function $f_{similarity}$ the user-defined similarity metric. For instance, when using Pearson's correlation coefficient [23], the fitness function is as follows:

$$f_{similarity} = \frac{\text{cov}(\Pi(C_{AVM_s}), \Pi(C_{AVM_t}))}{\sigma(\Pi(C_{AVM_s})) \cdot \sigma(\Pi(C_{AVM_t}))}, \tag{9}$$

where cov is the covariance between the variant attribute values $\Pi(C)$ of the source and the target AVM, and σ the standard deviation of the attribute values of all variants of the respective AVM.

4 EVALUATION: APPLICABILITY

The applicability of Loki depends on whether we can mimic actual changes in environments. That is, we need to evaluate whether a generated target AVM assembles changes to be useful in the evaluation of a transfer-learning technique. To this end, we state the following question:

RQ: With increasing user-defined degree of change, does Loki synthesize target AVMs that are harder to infer with transfer learning?

The rationale for this research question is that small changes and closely related target AVMs should be easy to learn with a transfer-learning approach than larger changes. If we observe better learnability of transfer learning for AVMs that are supposed to be very similar than those that are supposed to be different, we conclude that Loki is able to generate models according to the given specification.

4.1 Setup

To answer our research question, we synthesize several target AVMs with varying degree of change, based on source AVMs of different real-world software systems. With the resulting pairs of source and target AVMs, we can then run the transfer-learning technique L2S [11] to obtain an estimate for the target AVMs. Then, we compare the attribute values of options and interactions of the true and estimated target AVM as described below. Overall, we examine

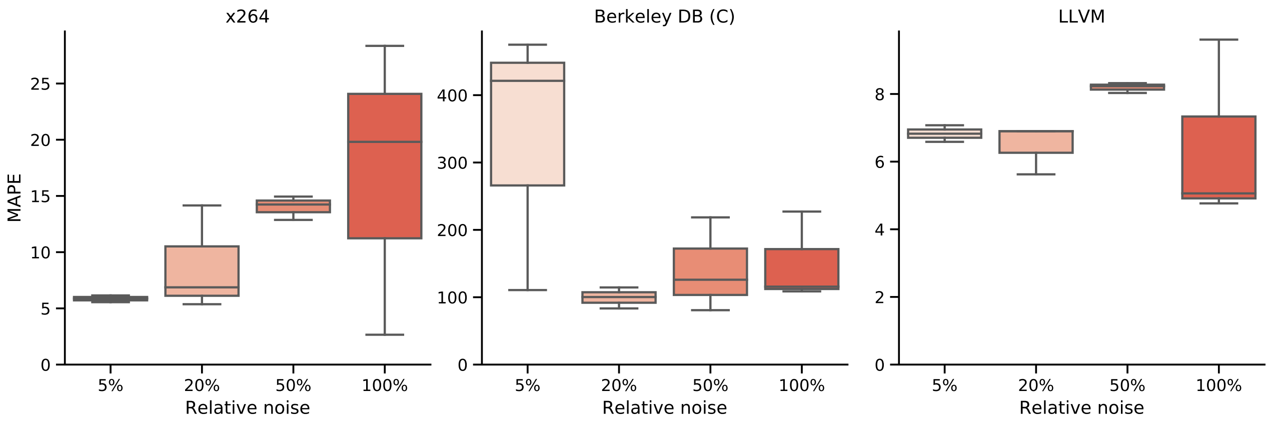

Figure 3: Mean absolute percentage error (MAPE) of learning-to-sample AVMs built on target AVMs with different noise levels.

whether the estimated AVMs are more accurate for target AVMs that were specified to be similar to the source AVMs than AVMs with a higher degree of change. In the following, we will describe our experiment in more detail.

There are several ways to specify degrees of change with Loki, as described in Section 3. Here, we opted for specifying different degrees of noise to specify the degree of change and keep other aspects constant. The rationale is that the noise can be highly controlled in our experiment and is simultaneously a stochastic operator. Moreover, noise can mimic also linear transfers (as we also observed during our experiments). The parameters were as follows:

- Loki changes attribute values of the top 50% influential options and interactions and
- maximizes Person's correlation coefficient for variant, option, and interaction attribute values of the source and target AVM.

We choose 5%, 20%, 50%, and 100% as relative noise levels for σ_{rel} in the noise specification for each target attribute value, which is computed according to Equation 5. Note that this equality will not strictly hold for every option and interaction in the target AVM, because we also account for the constant aspects.

As source AVMs, we rely on attribute measurements for each possible variant of three real-world software systems:

Berkeley DB (C) is an embedded database engine. The response time as attribute in the source AVM has been measured for different read and write queries.

X 264 is a video encoder for the H.264 compression format. From different metrics that were recorded during the encoding of a video clip, we use energy consumption as the attribute.

LLVM is a compiler infrastructure. With a compiling benchmark as workload, we use energy consumption as the attribute from several recorded metrics.

We refer to Kaltenecker et al. for details on the actual measurements [13]. For each subject system, we obtain a source AVM with Lasso regression using all available measurements. We use Loki to synthesize target AVMs with each noise level, for each source AVM. Finally, we run L2S on each pair of source and target AVM

Table 1: Mean absolute percentage error (MAPE) of AVMs built with L2S on different system's target AVMs with different noise levels.

	Noise			
	5%	20%	50%	100%
Berkeley DB (C)	421.3	100.3	126.0	115.7
LLVM	6.8	6.9	8.2	5.1
x264	5.9	6.9	14.2	19.8

and compute the accuracy for the resulting L2S AVMs [11]. The accuracy computation is provided by L2S; it computes the mean absolute percentage error (MAPE) for its evaluation set of n variants with Equation 10:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{\Pi(C^i_{AVM_t}) - \Pi(C^i_{\text{L2S}})}{\Pi(C^i_{AVM_t})} \right| \tag{10}$$

$\Pi(C^i_{AVM_t})$ is the measured variant attribute value of the target AVM, whereas the prediction of the L2S AVM is given as $\Pi(C^i_{\text{L2S}})$. We repeated each run three times and report the result MAPE as box plots.

4.2 Results

We report the median MAPE values for the three repetitions in Table 1 and visualize all MAPE values in Figure 3. Note that we use a training set size of only 5 configurations with which L2S infers a new AVM from a given model. This small number of training configurations led to acceptable results in prior tests and resembles a case of scarce data in the source environment. In what follows, we describe the results per subject system. For x264, the error increases with higher relative noise. A relative noise of 5% results in an easy-to-learn model. As expected, increasing noise (again, stochastically distributed among options and interactions) increases the difficulty to infer new attribute values that correspond to the ground truth (an increasing error of up to 20%).

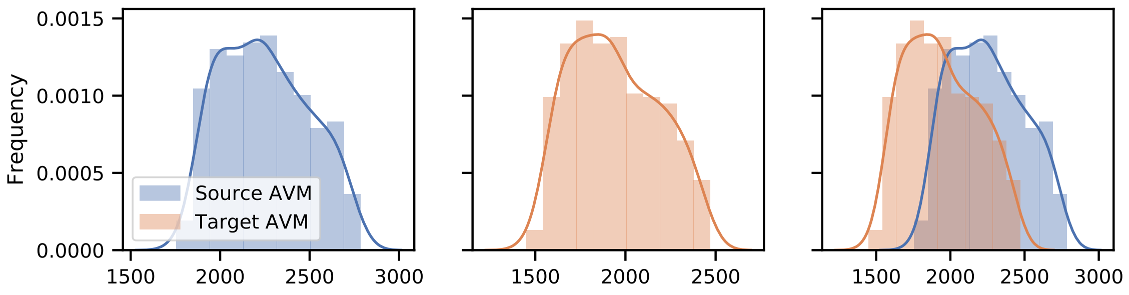

Figure 4: Variant attribute distributions of a source and a target AVM of LLVM with 20% relative noise.

The picture is different for Berkeley DB (C): for 5% noise, L2S results in an error rate of up to 400%. For the other noise levels, the error is still high. Although these results seem odd at first, they make actually sense when considering the low training set size of 5 configurations and the usage of three repetitions. We will discuss this aspect in more detail in Section 4.3. For LLVM, L2S achieves similar error rates for all noise levels; it has no difficulties in learning new attribute values.

4.3 Discussion

Berkeley DB (C). We obtained the highest MAPE for Berkeley DB (C) and error rates of up to 400% even when adding only 5% noise. We hypothesize that the low number of configurations as input for L2S might be a reason for the problems of inferring a correct model. Also, the variance in the selection of configurations for the training set seems to be more influential for the error rate than the noise level. To verify our assumptions, we conducted a second experiment, in which we increased the training set size to ten, which still are less than 0.4% of all valid configurations. Figure 5 shows the results. The error substantially decreased to a minimum of 5% for the 5% noise case. Also, the MAPE shows now a similar behavior as for x264.

x264. The results for x264 clearly meet our expectations in that a larger change to the source AVM results in more difficulties for learning a target AVM. The rationale is that L2S would need to see more configurations to account for the severe changes and that less information could be transferred from the source to the target. Here lies the power of Loki: We can perform fine-grained analysis to which degree information translation is possible for AVMs, which aspects are easier to transfer than others, and what configurations should be selected first by the transfer-learning approach to transfer information (i.e., attribute values, distributions, etc.) in the most efficient way.

LLVM. Since LLVM's average MAPE values of all noise levels are similar, we reason that source and target AVMs must be equally closely related for all noise levels. Indeed, reviewing Loki's output plots, we find that the target AVM's distribution of variant attribute values is usually only a shifted version of the source AVM' distribution. Figure 4 shows source and target variant attribute distributions for one replication for the 20% noise specification. A shift of the distribution of variant attribute values may happen when the option with the highest attribute value overshadows the influence of all other options on the attribute, thus shifting the whole distribution by the extent this option changes. A shift in the distribution of attribute values of variants resembles a scenario that has been found to occur for real-world systems as described in Section 2.1— an indication for Loki's applicability. Since L2S has been developed to utilize such environment changes, it is no surprise but to be expected that it achieves a high accuracy. This further backs our approach.

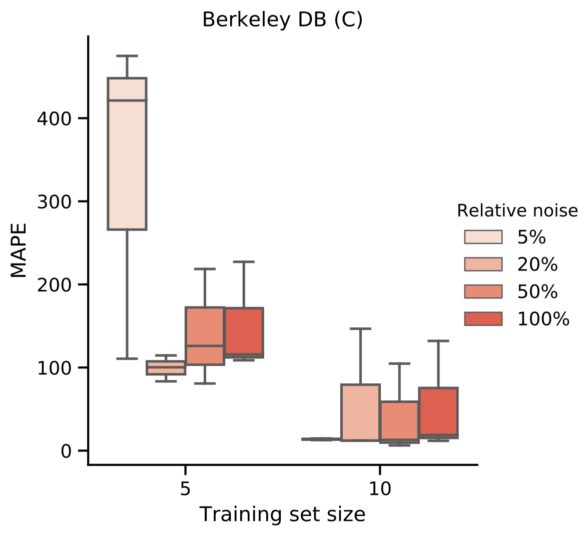

Figure 5: MAPE of L2S AVMs built on Berkeley DB (C) target AVMs with varying noise levels and training set sizes.

Our results show that Loki is indeed able to provide a test bed for transfer-learning techniques. We could replicate the error rates for L2S as expected: A higher noise level leads to more difficulties to learn a model. More importantly, we could even determine an optimal training set size for Berkeley DB (C) and demonstrate that the nature of the variability model together with the ratio of attribute values (very small to large values for Berkeley DB (C)) can influence the accuracy of transfer-learning techniques. Already this insight can stimulate new developments in this area. It seems that the training set size should be a function of the distribution and range of attribute values, and thus be determined per subject system. This agrees with the findings of LLVM, where even Gaussian changes result in large shifts of the whole value distribution. Hence, we answer our research question positively.

5 SUMMARY

Transfer learning is a growing research area in the field of configurable software systems, yet there exists no user-friendly tool that can synthesize target attributed variability models (AVMs) for a given source AVM such that both mimic a realistic environment change. For that reason, we propose Loki, a tool and approach to generate a target AVM such that it allows for simulating several aspects of environment changes at once.

Loki provides a user-friendly implementation leveraging a genetic algorithm to generate a target AVM according to multiple user-specified criteria. We evaluated Loki by generating multiple new AVMs from AVMs that have been determined from real-world subject systems. We assessed whether the existing transfer-learning approach *Learn to Sample* is able to infer the new attribute values depending on the degree of change we have performed. We found that the generated models indeed increase complexity for learning the changed attribute values depending on the degree of change. We also found that Loki can be used to find optimal training set sizes for different systems.

By making the tool and implementation publicly available, we aim at stimulating new research in this area and reducing the burden of conducting a full-fledged evaluation or parameter adjustment of a learning technique.

ACKNOWLEDGMENT

Siegmund's and Apel's work has been funded by the German Research Foundation (SI 2171/3-1, SI 2171/2, and AP 206/11-1).

REFERENCES

[1] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Software Product Lines*, Henk Obbink and Klaus Pohl (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 7–20. https://doi.org/10.1007/11554844_3

[2] Thomas M. Cover and Joy A. Thomas. 2012. *Elements of Information Theory*. John Wiley & Sons.

[3] W.W. Daniel. 1990. *Applied Nonparametric Statistics*. PWS-Kent Publ.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *Trans. Evol. Comp* 6, 2 (April 2002), 182–197. https://doi.org/10.1109/4235.996017

[5] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. [n. d.]. Variability-Aware Performance Modeling: A Statistical Learning Approach. 12.

[6] Huong Ha and Hongyu Zhang. 2019. DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network. *Proceedings - International Conference on Software Engineering* (May 2019), 1095–1106. https://doi.org/10.1109/ICSE.2019.00113

[7] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. 517–528. https://doi.org/10.1109/ICSE.2015.69

[8] Robert M. Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. 2016. SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization. *ACM Transactions on Software Engineering and Methodology* 25, 2 (April 2016), 1–39. https://doi.org/10.1145/2897760

[9] Md Shahriar Iqbal, Lars Kotthoff, and Pooyan Jamshidi. 2019. Transfer Learning for Performance Modeling of Deep Neural Network Systems. *arXiv:1904.02838 [cs]* (April 2019). arXiv:cs/1904.02838

[10] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. *arXiv:1709.02280 [cs, stat]* (Sept. 2017). arXiv:cs, stat/1709.02280

[11] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to Sample: Exploiting Similarities Across Environments to Learn Performance Models for Configurable Systems. (2018), 12.

[12] Mohammad Ali Javidian, Pooyan Jamshidi, and Marco Valtorta. 2019. Transfer Learning for Performance Modeling of Configurable Systems: A Causal Analysis. *arXiv:1902.10119 [cs]* (Feb. 2019). arXiv:cs/1902.10119

[13] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-Based Sampling of Software Configuration Spaces. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. 1084–1094. https://doi.org/10.1109/ICSE.2019.00112

[14] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *Ann. Math. Statist.* 22, 1 (March 1951), 79–86. https://doi.org/10.1214/aoms/1177729694

[15] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-Optimal Configurations in Product Lines by Random Sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*. 61–71. https://doi.org/10.1145/3106237.3106273

[16] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of Exact and Approximate Multi-Objective Optimization for Software Product Lines. In *Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC '14)*. 92–101. https://doi.org/10.1145/2648511.2648521

[17] Juliana Alves Pereira, Hugo Martin, Mathieu Acher, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2019. Learning Software Configuration Spaces: A Systematic Literature Review. *ArXiv* abs/1906.03018 (2019). arXiv:1906.03018

[18] Sergio Segura, José A. Galindo, David Benavides, José A. Parejo, and Antonio Ruiz-Cortés. 2012. BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12*. 63–71. https://doi.org/10.1145/2110147.2110155

[19] Kai Shi. 2017. Combining Evolutionary Algorithms with Constraint Solving for Configuration Optimization. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 665–669. https://doi.org/10.1109/ICSME.2017.32

[20] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-Influence Models for Highly Configurable Systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM, 284–294. https://doi.org/10.1145/2786805.2786845

[21] Norbert Siegmund, Stefan Sobernig, and Sven Apel. 2017. Attributed Variability Models: Outside the Comfort Zone. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 268–278. https://doi.org/10.1145/3106237.3106251

[22] C. Spearman. 1904. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology* 15, 1 (1904), 72–101.

[23] Student. 1908. Probable Error of Correlation Coefficient. *Biometrika* 6, 2-3 (Sept. 1908), 302–310. https://doi.org/10.1093/biomet/6.2-3.302

[24] Tian Huat Tan, Yinxing Xue, Manman Chen, Jun Sun, Yang Liu, and Jin Song Dong. 2015. Optimizing Selection of Competing Features via Feedback-Directed Evolutionary Algorithms. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. ACM, 246–256. https://doi.org/10.1145/2771783.2771808

[25] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17*. ACM Press, 39–50. https://doi.org/10.1145/3030207.3030216

[26] Fan Wu, Westley Weimer, Mark Harman, Yue Jia, and Jens Krinke. 2015. Deep Parameter Optimisation. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. ACM, 1375–1382. https://doi.org/10.1145/2739480.2754648

[27] Yi Xiang, Yuren Zhou, Zibin Zheng, and Miqing Li. 2018. Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers. *ACM Transactions on Software Engineering and Methodology* 26, 4 (Feb. 2018), 1–46. https://doi.org/10.1145/3176644