Adressing the Impact of Workload Variation on the Performance of Configurable Software Systems

Abstract—**The performance characteristics of a software system depends to a significant extent on its configuration and workload. State-of-the-art performance modeling approaches either address configuration-dependent or workload-dependent performance behavior. The interaction of both factors and how they influence performance have not been systematically studied so far. Understanding to what extent configuration and workload—individually and combined—cause a software system's performance to vary is key to understand whether performance models are generalizable, across different configurations and workloads. Assessing the impact and driving factors of such input sensitivity is key to develop strategies that obtain representative performance prediction models.**

To shed light on this issue, we have conducted a *systematic* empirical study, analyzing a multitude of configurations and workloads across a six software systems. We have obtained a substantial number of black-box performance measurements and enriched them with coverage data to assess whether and how configuration choices and workloads interact and shape software performance. We find that code coverage (i.e., *what* code is executed) and code utilization (i.e., *how* covered code is executed) are driving factors for workload-specific performance differences. Beyond code coverage testing, our findings motivate the use of dynamic code analyses to identify whether and in which way configuration options are sensitive to varying the workloads.

I. Introduction

Most modern software systems can be customized via configuration options to meet user demands. Configuration options can enable desired functionality or tweak non-functional aspects of a software system, such as improving performance or energy consumption. The relationship of configuration choices and their influence on performance has been extensively studied in the literature [Dorn, Apel, and SiegmundDorn et al.2020], [Siegmund, Grebhahn, Apel, and KästnerSiegmund et al.2015], [Ha and ZhangHa and Zhang2019a], [Shu, Sui, Zhang, and XuShu et al.2020], [Guo, Czarnecki, Apel, Siegmund, and WasowskiGuo et al.2013], [Sarkar, Guo, Siegmund, Apel, and CzarneckiSarkar et al.2015], [Guo, Yang, Siegmund, Apel, Sarkar, Valov, Czarnecki, Wasowski, and YuGuo et al.2018], [Zhang, Guo, Blais, and CzarneckiZhang et al.2015], [Ha and ZhangHa and Zhang2019b]. The backbone of performance estimation are prediction models that map a given configuration to the estimated performance value. Learning performance models relies on a training set of configuration-specific performance measurements. In state-of-the-art approaches observations usually employ only a single workload which aims at emulating a specific real-world application scenario.

The choice of the workload (i.e., the input fed to the software system) is known to influence the performance of configurable software systems in different ways as has been shown for the domains of SAT solvers [Falkner, Lindauer, and HutterFalkner et al.2015], [Xu, Hutter, Hoos, and Leyton-BrownXu et al.2008], compilation [Ding, Ansel, Veeramachaneni, Shen, O'Reilly, and Amarasin [Plotnikov, Melnik, Vardanyan, Buchatskiy, Zhuykov, and LeePlotnikov e video transcoding [Maxiaguine, Liu, Chakraborty, and OoiMaxiaguine et a [Pereira, Acher, Martin, and JézéquelPereira et al.2020], data compression [Khavari Tavana, Sun, Bohm Agostini, and KaeliKhavar and code verification [Koc, Mordahl, Wei, Foster, and PorterKoc et al.202 Besides apparent interactions, such as performance scaling with the size of a workload, qualitative aspects can result in more complex and non-trivial performance interactions. Take as an example the distributions of configuration-specific throughput of the database h2 in Figure 1. Here, we tested the exact same configurations on two different parameterizations of the benchmark TPC-C. The scale factor controlled the complexity (number of entities modeled) of the benchmark. While for most configurations, throughput decreases for a more complex benchmark, some configurations achieve higher throughput for a more complex benchmark. A similar example was outlined by Pereira et al. for the video encoder x264. That is, configuration-specific performance can be highly sensitive to workload variation and the behavior under different workloads can change in unforeseeable ways. In turn, this can render performance models based on a single workload useless, unless configuration options' sensitivity to workloads is accounted for.

To address this limitation, two different directions have been pursued in the literature. First, performance models trained using a specific workload can be adapted to another specific workload. Second, one can specify workload characteristics as further independent variables

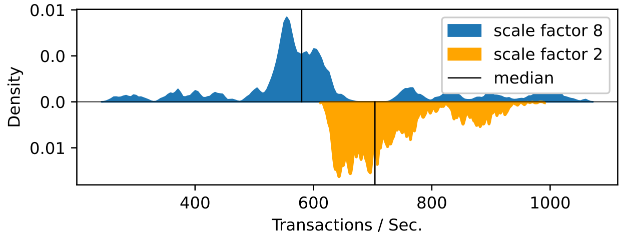


Figure 1: Performance distributions of the database system h2 run the TPC-C benchmark at different scale factors.

when modeling configuration-dependent performance [Koc, Mordahl, Wei, Foster, and PorterKoc et al.2021]. The first strategy direction relies on transfer learning techniques, where, given an existing performance model, in a separate step only the differences to a new environment are learned. Such a transfer function encodes which configuration options' influence on performance is sensitive to workload variation. While transfer learning is an effective strategy that is not limited to varying workloads [Jamshidi, Velez, Kästner, and SiegmundJamshidi et al.2018], but can also be applied to different versions [Jamshidi, Siegmund, Velez, Kästner, Patel, and AgarwalJamshidi et al.2017a], [Jamshidi, Velez, Kästner, Siegmund, and KawthekarJamshidi et al.2017b], [Martin, Acher, Lesoil, Jezequel, Khelladi, and PereiraMartin et al.2021], or hardware setups [Ding, Pervaiz, Krishnan, and HoffmannDing et al.2020], its main limitation is that the transfer function is specific to the differences between two environments.

In contrast to transfer learning, a more generalist approach is to consider the input fed to a software system as a further dimension for modeling performance. Here, a workload can be characterized by properties that—individually or in conjunction with software configuration options—influence performance. For such a strategy to work, one requires knowledge of the characteristics of a workload that influence performance. This strategy has been effectively tested for a variety of application-domains, such as program verification. However, the added complexity comes at significant cost. Not only does this require substantially more measurements, we often lack knowledge of which performance-relevant characteristics best describe workloads.

The existing body of research reflects the prevalence and importance of the workload influence on software systems. All these works are aware of the workload dimension as a factor of performance variation, yet little is known about the quality and driving factors of the *interplay* between configuration options and workloads. Our understanding of this cross-factor relationship lacks knowledge of the following aspects:

— How different is configuration-specific performance across different workloads?
— How many configuration options are responsible for differences in workload-specific performance behavior?
— What are the driving factors of the interplay between configuration options and workloads with regard to performance?

To answer these questions, we have conducted a systematic empirical study that sheds light on whether and how configuration options and workload choices interact with regard to performance. Specifically, we analyze 29 347 configurations and 55 workloads across six configurable software systems to obtain a broad picture of the interaction of configuration and workload when learning performance models and estimating a configuration's performance (i.e., response time). Aside from studying the sole effects of workload variation on performance behavior, we explore possible driving factors. To this end, we enrich performance observations with corresponding statement coverage data to understand workload variation at finer granularity.

Our findings show that varying the workload can influence configuration-dependent software performance in different ways, including non-linear and non-monotonous effects. Our findings suggest that (a) coverage of code specific to configuration options as well as (b) how such code is utilized are driving factors of input sensitivity. A key insight is that, to maintain and improve performance model representativeness, an additional notion of input sensitivity has to be considered. We argue that the use of code analysis techniques to address input sensitivity when varying the workload and maintain and improve the representativeness of a performance-prediction model.

To summarize, we make the following contributions:

— An empirical study of 29 347 configurations and 55 workloads across six configurable software systems on whether interactions of workloads with configuration options affect performance and what factors can drive such interactions;
— A detailed analysis that illustrates that variation in code coverage and code utilization due to varying workloads can affect the influence of configuration options on software performance;
— A companion Web site[1] with supplementary material including performance and coverage measurements, experiment workloads and configurations, and an interactive dashboard[2] for additional visualizations left out due to space limitations.

II. Background and Problem Statement

A. Performance Prediction Models

Configurable software systems are an umbrella term for any kind of software system that exhibits configuration options to customize functionality. While the primary purpose of configuration options is to select and tune functionality, each configuration choice may also have implications on

[1] https://github.com/fse-submission-2022/workload-performance/
[2] https://workload-performance.herokuapp.com/

non-functional properties—be it intentional or unintentional. There are different approaches to capture the relationship between configuration options and performance indicators, most basically either *analytical* or *empirical* in nature. All share the objective to approximate non-functional properties, such as execution time or memory usage, as a function of software configurations $c \in C$, formally $\Pi : C \rightarrow R$.

Analytic models incorporate existing knowledge about the operations of a software system, comparable to estimating an algorithm's complexity [Brown, Falgout, Jones, Jim, and JonesBrown et al.2000], [Gahvari, Baker, Schulz, Yang, Jordan, and GroppGahvari et al.2011]. Here, one deliberately includes or excludes configuration options as predictors and selects a model structure following the current understanding of the software system. While it avoids ambiguity in terms of feature selection and explainability, analytic approaches do not guarantee to cover unanticipated idiosyncrasies or interactions between configuration options.

Empirical performance models, by contrast, do not rely on an understanding of the software system, but on a set of configuration-specific observations. In this vein, finding configurations with optimal performance [Nair, Menzies, Siegmund, and ApelNair et al.2017], [Nair, Yu, Menzies, Siegmund, and ApelNair et al.2020], [Oh, Batory, Myers, and SiegmundOh et al.2017] and estimating the performance for arbitrary configurations of the configuration space is an established line of research [Siegmund, Grebhahn, Apel, and KästnerSiegmund et al.2015], [Ha and ZhangHa and Zhang2019a], [Shu, Sui, Zhang, and XuShu et al.2020], [Guo, Czarnecki, Apel, Siegmund, and WasowskiGuo et al.2013], [Sarkar, Guo, Siegmund, Apel, and CzarneckiSarkar et al.2015], [Guo, Yang, Siegmund, Apel, Sarkar, Valov, Czarnecki, Wasowski, and YuGuo et al.2018], [Zhang, Guo, Blais, and CzarneckiZhang et al.2015], [Ha and ZhangHa and Zhang2019b]. Empirical performance models can be obtained using a variety of machine-learning techniques, including probabilistic programming [Dorn, Apel, and SiegmundDorn et al.2020], multiple linear regression [Siegmund, Grebhahn, Apel, and KästnerSiegmund et al.2015], classification and regression trees [Guo, Czarnecki, Apel, Siegmund, and WasowskiGuo et al.2013], [Sarkar, Guo, Siegmund, Apel, and CzarneckiSarkar et al.2015], [Guo, Yang, Siegmund, Apel, Sarkar, Valov, Czarnecki, Wasowski, and YuGuo et al.2018], Fourier learning [Zhang, Guo, Blais, and CzarneckiZhang et al.2015], [Ha and ZhangHa and Zhang2019b], and deep neural networks [Ha and ZhangHa and Zhang2019a], [Shu, Sui, Zhang, and XuShu et al.2020].

The set of configurations for training can be sampled from the configuration space using a variety of different sampling techniques [Kaltenecker, Grebhahn, Siegmund, and Ape All sampling strategies aim at yielding a representative sample, either by covering the main effects of configuration options and interactions among them [Siegmund, Kolesnikov, Kästner, Apel, Batory, Ros or sampling uniformly from the configuration space [Oh, Batory, Myers, and SiegmundOh et al.2017], [Kaltenecker, Grebhahn, Siegmund, Guo, and ApelKaltenecker et al.2019 Most approaches share the perspective of treating a configurable software system as a black-box model at application-level granularity. Recent work has incorporated feature location techniques to guide sampling effort towards relevant configuration options [Velez, Jamshidi, Sattler, Siegmund, Apel, a [Velez, Jamshidi, Siegmund, Apel, and KästnerVelez et al.2021] or model non-functional properties at finer granularity [Weber, Apel, and SiegmundWeber et al.2021].

B. Varying Workloads

When assessing the performance of a software system, we ask how well a certain *operation* is executed, or, phrased differently, how well an *input fed to the software system* is processed. Such inputs, commonly called workloads, are essential to assessing performance, even detached from the specific context of configurable software systems. By nature, the workload of a software system is application-specific, such as a series of queries and transactions fed to a database system, a set of raw image files for video encoding, or an arbitrary file for data compression etc. Workloads can often be distinguished by characteristics (dimensions) they exhibit, such as their file size in general, or, the type of data to be compressed (text, binary data) for instance. A useful workload for assessing performance or benchmarking should, in practice, closely represent the real-world scenario that the system under test is deployed in. To achieve this, a well-defined and widely employed technique in performance engineering is workload characterization [Ceesay, Lin, and BarkerCeesay et al.2020], [Papadopoulos, Versluis, Bauer, Herbst, Kistowski, Ali-Eldin, Abad, Ama To select a representative workload, it is imperative to explore workload characteristics and validate a workload with real-world observations. This can be achieved by constructing workloads, among others, from usage patterns [Calzarossa, Massari, and TesseraCalzarossa et al.2016], or by increasing the workload coverage by using a mix of different workloads rather than a single one [Jiang and HassanJiang and Hassan2015].

While workload characterization and benchmark construction is domain-specific, there are numerous examples of this task being driven by community efforts instead of individuals. For instance, the non-profit organizations Standard Performance Evaluation Corporation (SPEC) and Transaction Processing Performance Council (TPC) provide large bodies of benchmarks for data-centric applications or across different domains, respectively.

C. Representative Estimations?

While the notion of representative workloads is ubiquitous in an industry setting, we face a different situation in research on empirical performance models (cf. Section II-A): Most approaches provide accurate performance estimations, yet are based on observations gained by varying configurations while keeping the workload the same. Clearly, this can limit the model's generalizability to different workloads, especially if the training workload poorly represents real-world scenarios. While the configuration-specific behavior might be congruent across different workloads (i.e, a configuration scoring comparably for different workloads), we cannot rely on such assumptions in practice, where different workloads can indeed result in entirely different configuration-specific performance behavior.

Revisiting an observation by Pereira et al. [Pereira, Acher, Martin, and JézéquelPereira et al.2020], the following example illustrates that even seemingly small differences in the composition of a workload can induce performance behavior that is hard to anticipate: We tested the performance (throughput: transactions per second) of a number of configurations for the database system h2 across two different workloads. Both workloads are instances of the database benchmark TPC-C, consisting of a fixed-ratio mix of transactions (inserts, updates, selects) for a specific database schema. The only difference was varying the scale factor, which controls for the number of warehouses modeled in the schema. The resulting performance distributions are given in Figure 1.

While one might first expect that a more complex workload results in lower throughput, this is indeed the case, but does not hold for all configurations. The median throughput decreases for the larger scale factor, yet the shape and spread of the distribution are entirely different. Most notably, the maximum throughput for the greater scale factor is higher than for the smaller scale factor, indicating that some configurations indeed do not follow the general trend. This illustrates that the effect of varying the workload for some configurations cannot be captured by a linear transformation (constant shift or scaling).

Some aspects of this *input sensitivity* have been observed and documented before in the literature [Liao, Chen, Li, Zeng, Shang, Guo, Sporea, Toma, and SajediLiao et al.2020], [Pereira, Acher, Martin, and JézéquelPereira et al.2020], [Jamshidi, Siegmund, Velez, Kästner, Patel, and AgarwalJamshidi et al.2017a] and raise questions, such as: *Which options are input sensitive? What are the driving factors for input sensitivity? Can we estimate which options are input-sensitive?* We set out to answer these questions in this paper.

D. Workloads and Performance Prediction

To some aspects of the said questions, different approaches have been proposed to tackle the problem of input sensitivity.

a) Workload-aware Performance Modeling: Extending on workload characterization (cf. Section II-B), a strategy that embraces workload diversity is to incorporate workload characteristics into the problem space of a performance prediction model. Here, performance is modeled as a function of both the configuration options exhibited by the software system as well as the workload characteristics, formally $\Pi : C \times W \to R$. The combined problem space enables learning performance models that generalize to workloads that exhibit characteristics denoted by W since we can screen for performance-relevant combinations of options and workload characteristics. Although this strategy is highly application-specific, it has been successfully applied to different domains, such as program verification [Koc, Mordahl, Wei, Foster, and PorterKoc et al.2021]. However, its main disadvantages are twofold: The combined problem space (configuration and workload dimension) requires substantially more observations to screen for identifying performance-relevant options, characteristics, and combinations thereof. In addition, previous work found that that only few configuration options are input sensitive [Jamshidi, Siegmund, Velez, Kästner, Patel, and A when varying the workload. That is, the problem of identifying meaningful, but sparse predictors is exacerbated since one must not only identify performance-relevant configuration options, but also input-sensitive ones.

b) Transfer Learning for Performance Models: Another strategy that builds on the fact that, across different workloads, only few configuration options are in fact input sensitive [Jamshidi, Siegmund, Velez, Kästner, Patel, and AgarwalJamshidi et al.2017a]. Here one first trains a model on a standard workload and, subsequently, adapts it to different workloads. Contrary to a generalizable workload-aware model, transfer learning strategies focus on approximating a transfer function that, without characterizing the workload, encodes the information of which configuration options are sensitive to differences between a source and target pair of workloads. Training a workload-specific model and adapting it on occasion provides an effective means to reuse performance models, which is not limited to workloads [Jamshidi, Velez, Kästner, and SiegmundJamshidi et al.2018], but has successfully been applied to different hardware setups [Ding, Pervaiz, Krishnan, and HoffmannDing et al.2020], [Valov, Petkovich, Guo, Fischmeister, and CzarneckiValov et al.2017] and across versions [Martin, Acher, Lesoil, Jezequel, Khelladi, and PereiraMartin et al.2021]. The main shortcoming of transfer learning approaches is that they do not generalize to arbitrary workloads, since a transfer function is tailored to a specific target workload. Here, one trades off generalizability and measurement cost as learning a transfer function requires substantially fewer training samples.

While both directions are effective means to handle input sensitivity, to the best of our knowledge, there is no *systematic* assessment of the factors that drive the interaction between configuration options and workloads with regard to performance. Understanding scenarios that are associated with or even cause incongruent performance influences across workloads can help practitioners to employ established analysis techniques more effectively and can motivate researchers to devise analysis techniques dedicated to such scenarios.

III. Study Design

In what follows, we describe the general experiment setup and study design as well as research questions. We make all performance measurement data, configurations, workloads, and learned performance models available on the paper's companion Web site.

A. Research Questions

Our first two research questions shed light on the input sensitivity of the performance behavior of the studied software systems. We first take a look at systems as a whole (RQ_1) with regard to a large set of configurations and, subsequently, consider individual configuration options (RQ_2). Extending on the results of RQ_2, we explore possible driving factors and indicators for workload-specific performance variation (RQ_3).

1) Performance Variation Across Workloads: Performance variation can arise from differences in the workload [Kounev, Lange, and von KistowskiKounev et al.2020]. In a practical setting, the question arises whether, and if so, to what extent an existing workload-specific performance model is representative of the performance behavior of other workloads. That is, can a model estimating performance of different configurations be reused for the same software system but run with a different workload? Depending on the degree of similarity of the performance behavior across workloads, we obtain a clearer picture of the prevalence of input sensitivity and to what extent the strategies outlined in Section II-D might be applicable. To this end, we formulate the following research question:

> RQ_1 *To what extent does performance behavior vary across workloads?*

2) Option Influence Across Workloads: At large, performance behavior is the resulting effect arising from multiple configuration options' and combinations' respective influences. To understand which configuration options are driving performance variation, in general, and which are input sensitive, in particular, we formulate the following research question:

> RQ_2 *To what extent do influences of individual configuration options depend on the workload?*

3) Causes of Input Sensitivity: The first two research questions describe the performance behavior of our subject systems: Based on the results of related work, we expect configuration options to be, at least, to some extent input sensitive. To contextualize our findings, we switch our perspective to the code level. The goal is to understand the relationship between input sensitivity (i.e., variation in the performance influence of configuration options) and the execution of the subject system under varying workloads. We hypothesize that executions under different workloads also exhibit variation with respect to what code sections are executed and how this code is used. Using code coverage analysis—an easy to understand and widely employed technique—we are interested in how far one could infer or explain performance influence variation just based on code.

> RQ_3 *Does the variation in configuration options' performance influence across workloads correlate with differences in the respective execution footprint?*

B. Experiment Setup

1) Subject System Selection: We selected twelve configurable software systems for our study. To ensure that our findings are not specific to one domain or ecosystem, the selection comprises an equal mix of Java and

Table I: Subject System Characteristics

System	Language	Application Type	Version	# O	# C	# W
jump3r	Java	Audio Encoder	1.0.4	19	4 196	6
kanzi	Java	File Compressor	1.9	24	4 112	9
dconvert	Java	Image Scaling	1.0.0-alpha7	17	6 764	12
h2	Java	Embedded Database	1.4.200	16	1 954	8
batik	Java	SVG Rasterizer	1.14	10	1 919	11
jadx	Java	Java Decompiler	1.2.0	18	10 502	9
xz	C/C++	File Compressor	5.2.0	33	1 898	13
lrzip	C/C++	File Compressor	0.651	11	190	13
z264	C/C++	Video Encoder	baee400...	–	–	–
z3	C/C++	SMT Solver	4.8.14	–	–	–

Abbreviations: # O: No. of options, # C: No. of configurations, # W: No of. workloads

C/C++ systems from different application domains (cf. Table I). We include systems studied in previous and related work [Velez, Jamshidi, Sattler, Siegmund, Apel, and KästnerVelez et al.2020] [Weber, Apel, and SiegmundWeber et al.2021], [Pereira, Acher, Martin, and JézéquelPereira et al.2020] and incorporate further ones with comparable size and configuration complexity. All systems operate by processing a domain-specific input fed to them (henceforth called *workload*). This study treats execution time as the key performance indicator with the exception of h2, where we report throughput.

2) Workload Selection: This study relies on a selection of workloads for each domain or software system. Ideally, each set of workloads is diverse enough to be representative of most possible use cases. We selected the workload sets in this spirit, but cannot always guarantee a measurable degree of diversity and representativeness. This is due to the opacity of workloads: Beyond educated guesses, prior to conducting measurements, it is not possible to state which workload characteristics (size, scale, file type etc.) are performance-relevant. We discuss this aspect in the threats to validity. Below we outline the twelve case studies along with the workloads tested.

— For the *audio encoder* jump3r, the measured task was to encode raw WAVE audio signals to MP3 (jump3r). We selected a number of different audio files from the Wikimedia Commons collection and aimed at varying the file size/signal length, sampling rate, and number of channels. Both applications share all workloads.

— For the *video encoder* z264, the measured task was to encode raw video frames (y4m format). We selected a number of files from xiph.org's "derf collection", a set of test media for a variety of use cases. The frame files vary in resolution (low/SD up to 4K) and file size. Both applications in this domain were tested with the same workload set.

— For the *file compression* tools kanzi, xz, and lrzip, we used a variety of community compression benchmarks that represent different goals, including mixes of files of different types (text, binary, structured data etc.) or single-type files. We augmented this set of workloads with custom data, such as the Hubble Deepfield image and a binary of the Linux kernel. Beyond this set of workloads, for xz and lrzip we added different parameterizations of the UIQ2 benchmark to study the effect of varying file size.

— For the *SMT solver* z3, the measured task was to decide the satisfiability (find a solution or counter example) of a range of logical problems expressed in the SMT2 format. We selected the six longest-running problem instances from z3's performance test suite and augmented it with additional instances from the SMT2-Lib repository to cover different types of logic and increase diversity.

— For the *SVG rasterizer* batik, the measured task was to transform a SVG vector graphic into a bitmap. We selected a number of resources from the Wikimedia Commons collection, primarily varying the file size.

— For the embedded *database* h2, we used a selection of four benchmarks (SmallBank, TPC-H, YCSB, Voter) from OLTPBench [Difallah, Pavlo, Curino, and Cudre-Mauro a load generator for databases that allows for using a variety of performance testing benchmarks. For each benchmark, we varied the scale factor, which controls the complexity (number of entities modeled) in each scenario.

— For the *Java decompiler* jadx, the measured task was to decompile a number of Android applications in DEX byte code. We selected a number of APK packages from APKMirror.com/ from different domains (social media, games, utility etc.) and of varying size.

— For the *image scaler* dconvert, the measured task was to transform resources (image files, Photoshop sketches) at different scales (useful for Android development). We selected files that reflect dconvert's documented input formats (JPEG, PNG, PSD, and SVG) and vary in file size.

3) Configuration Sampling: For each subject system, we sampled a set of configurations. As exhaustive coverage of the configuration space is infeasible due to combinatorial explosion [Henard, Papadakis, Harman, and Le TraonHenar for binary configuration options, we combine several coverage-based sampling strategies and uniform random sampling into an *ensemble* approach: We employ option-wise and negative option-wise sampling [Siegmund, Grebhahn, Apel, and KästnerSiegmund et al.20 where each option is enabled once (i.e., in, at least, one configuration), or all except one, respectively. In addition, we use pairwise sampling, where two-way combinations of configuration op-

Table II: Hardware specifications of the compute clusters used.

Cluster	CPU	Clock	RAM	OS	Kernel
I	Intel Xeon E5-2630v4	2.2 GHz	256 GB	Debian 10	4.19.0-17
II	Intel Core i7-8559U	2.7 GHz	32 GB	Debian 10	4.19.0-14
III	Intel Core i5-8259U	2.3 GHz	32 GB	Debian 10	4.19.0-14

tions are systematically selected. Interactions of higher degree could be found accordingly, however, it is computationally prohibitively expensive [Henard, Papadakis, Harman, and Le TraonHenard et al.2015]. Last, we augment our sample set with a random sample that is, at least, the size of the coverage-based sample. To achieve a nearly uniform random sample, we used *distance-based sampling* [Kaltenecker, Grebhahn, Siegmund, Guo, and ApelKaltenecker et al.2019]. If a software system exhibited numeric configuration options, we varied them across, at least, two levels to account for their effect.

4) Coverage Profiling: To assess what lines of code are executed for each combination of workload and software configuration, we used two separate approaches for Java and C/C++. For Java, we used the on-the-fly profiler JACOCO[3] that intercepts byte code running on the JVM at run-time. For C/C++, we added instrumentation code to the software systems using Clang/LLVM to collect coverage information. In both cases, we split the performance measurement and coverage analysis runs to avoid distortion from the profiling and instrumentation overhead.

5) Measurement Setup: All experiments were conducted on three different compute clusters (cf. Table II), where all machines within a compute cluster had the identical hardware setup. All clusters ran a headless Debian installation. To minimize measurement noise, we used a controlled environment, where no additional user processes were running in the background, and no other than necessary packages were installed. We ran each subject system exclusively on a single cluster: h2 on cluster I; dconvert, batik and jadx on cluster II; the remaining systems on cluster III.

For all data points, we report the median across five repetitions (except for h2), which has shown to be a good trade-off between variance and measurement effort. For h2, we omitted the repetitions as, in a pre-study, running on the identical cluster setup, we found that across all benchmarks the coefficient of variation of the throughput was consistently below 5 %.

IV. Study Results

A. Comparing Performance Distributions (RQ_1)

1) Operationalization: We answer RQ_1 by pairwisely comparing the performance distributions from different workloads (cf. the comparison in Figure 1) and by determining whether any two distributions are similar or, if not, can be transformed into each other. For the latter case, we are specifically interested in what type of transformation is necessary as this determines *how* complex a workload interacts with configuration options. Specifically, we categorize each pair of workloads with respect to the following aspects:

1) **Similarity**: To test whether workload variation has any effect at all, we employ statistical tests. We use the non-parametric Wilcoxon signed-rank test [LovricLovric2010] because performance distributions are often multi-modal or long-tailed [Curtsinger and BergerCurtsinger and Berger2013], [Maricq, Duplyakin, Jimenez, Maltzahn, Stutsman, and RicciMaricq failing to meet requirements for parametric methods. We assess whether varying the workload affects configuration-specific performance. If we can reject the null hypothesis H_0 at $\alpha = 0.95$, we consider the distributions dissimilar. To account for overpowering due to high and different sample sizes (cf. Table I), we further report effect sizes to weed out negligible effects. Following the interpretation guidelines from Romano et al. [Romano, Kromrey, Coraggio, Skowronek, and DevineRomano et we use Cliff's δ [CliffCliff1993] and a threshold effect size of $|\delta > 0.147|$.
2) **Linear Correlation**: To test whether both performance distributions are shifted by a constant value or scaled by a constant factor, we compute for each pair of distributions Pearson's correlation coefficient r. To discard the sign of relationship, we use the absolute value and consider $|r| > 0.6$ indicates a strong linear relationship.
3) **Monotone Correlation**: Finally, we test whether there exists a monotonous relationship between the two performance distributions. We use Kendall's rank correlation coefficient τ [KendallKendall1938] and consider $|\tau| > 0.6$ a strong monotonous relationship.

Based on these three tests and metrics, we composed four categories that each pair of performance distributions can be categorized into. If we cannot reject H_0, we consider them identical and as similar distributions (SD). If both distributions exhibit a strong linear relationship, we classify them as linearly transformable (LT). If we observe a strong monotonous, but not a linear relationship, we classify such pairs as exclusively monotonously transformable into a separate category (XMT). Last, if the comparison yields no monotonous relationship, we can only transform them using non-monotonous methods (NMT). We summarize the category criteria as well as the category counts in Table III.

2) Results: We summarize the results of our classification in Table IV. For all of the six software systems, vary-

[3] https://www.jacoco.org/jacoco/trunk/doc/

Table III: Four disjoint categories of relationships between pairs of workload-specific performance distributions and their respective criteria.

Abbrev.	Category	Criteria
SD	Statistically similar distributions	H_0 not rejected and $\delta > 0.147$
LT	Strictly linear transformation	$r^* \geq 0.6$
XMT	Non-linear, monotonous transformation	$r^* < 0.6$ and $\tau^* \geq 0.6$
NMT	Non-monotonous transformation	(otherwise)

Table IV: Frequency of each category (cf. Table III) for each software system studied.

Subject System	SD		LT		XMT		NMT	
	abs	*rel*	*abs*	*rel*	*abs*	*rel*	*abs*	*rel*
jump3r	0	0 %	15	100.0 %	0	0 %	0	0 %
kanzi	0	0 %	28	77.8 %	4	11.1 %	4	11.1 %
dconvert	0	0 %	29	43.9 %	0	0 %	37	56.1 %
h2	0	0 %	13	46.4 %	0	0 %	15	53.6 %
batik	0	0 %	28	50.9 %	8	14.6 %	19	34.6 %
jadx	0	0 %	120	100.0 %	0	0 %	0	0 %
xz	0	0 %	64	82.0 %	1	1.3 %	13	16.7 %
lrzip	0	0 %	57	73.0 %	13	16.7 %	8	10.3 %
z264	0	0 %	0	0 %	0	0 %	0	0 %
z3	0	0 %	0	0 %	0	0 %	0	0 %

ing the workloads has a noticable effect on the performance distribution. All software systems, at least, in part, exhibit performance distributions which can be transformed into one another using an linear transformation, such as shifting by a constant value or scaling by a constant factor. In particular, for jump3r and jadx, we did observe only such behavior. This finding corresponds to experimental insights from Jamshidi et al., who encoded differences between performance distributions using linear functions [Jamshidi, Siegmund, Velez, Kästner, Patel, and AgarwalJamshidi et al.2017a]. For four software systems, we obtained a more diverse picture: For kanzi and batik, a few performance distributions require transformations that are non-linear, but still monotonous. For dconvert and h2, the majority of performance distribution pairs cannot be described by a monotonous relationship. In total, four out of the six software systems exhibit *non-monotonous* relationships across, at least, one workload.

The observed range of relationship types across six software systems had no type prevail across all software systems. The large number of linear relationships suggests that varying the workload does not pose a general obstacle to learning or adapting performance models However, the presence of non-monotonous relationships besides other ones emphasizes that (a) there exist indeed cases where adapting performance models across workloads is challenging. That is, adressing and handling non-monotinicity require more information about what factors (workload characteristics and configuration options) are driving workload-dependent effects.

Summary (RQ_1): Varying the workload causes a substantial amount of variation among performance distributions. Across workloads, we observed *mostly linear*, but to a large extent, also *non-monotonous* differences.

B. Input Sensitivity of Options (RQ_2)

1) Operationalization: To address RQ_2, we need to determine the configuration options' influence on performance and assess their variation across workloads.

Explanatory Model: To obtain accurate and interpretable performance influences per option, we learn an explanatory performance model using the entire sample set that is based on multiple linear regression [Dorn, Apel, and SiegmundDorn et al.2020], [Siegmund, Grebhahn, Apel, and KästnerSiegmund et al.2015], [Ha and ZhangHa and Zhang2019b]. Here, each variable in the linear model corresponds to an option and each coefficient represents the corresponding option's influence on performance. We limit the set of independent variables to individual options rather than including higher-order interactions to be consistent with the feature location used for RQ_3 where we determine option-specific, yet not interaction-specific code segments.

Standardization: To facilitate the comparison of regression coefficients across workloads, we follow common practice in machine learning and standardize our dependent variable by subtracting the population's mean performance and divide the result by the respective standard deviation. Henceforth, we refer to these standardized regression coefficients as *relative performance influences*. A beneficial side effect of standardization is that the observed variation of regression coefficients for each configuration option cannot be attributed to shifting or scaling effects (affine transformation, class LT in Table III). This way, we can pin down the non-linear or explicitly non-monotonous effect that workloads may exercise on performance.

Handling Multicollinearity: Multicollinearity is a standard problem in statistics and emerges when features are correlated [DaoudDaoud2017]. This can, for instance, arise from groups of mutually exclusive configuration options and result in distorted regression coefficients [Dorn, Apel, and SiegmundDorn et al.2020]. Although the model's prediction accuracy remains unaffected, we cannot trust and interpret the calculated coefficients. To mitigate this problem and, in particular, to ensure that the obtained performance influences remain interpretable, we follow best practices and remove specific configuration options from the sample that cause multicollinearity [Dorn, Apel, and SiegmundDorn et al.2020]. For the training step, we exclude all mandatory configuration options since these, by definition, cannot contribute to performance variation. In addition, for each group of mutually exclusive configuration options, we discard one group member. These measures reduced the variance inflation factor (indicating multicollinearity) to a negligible degree [O'BrienO'Brien2007].

From the comparison of the relative performance influences, we can answer RQ_2 in detail and assess how many configurations are sensitive to varying the workload, what characteristic traits describe the performance influences, and whether we can identify patterns.

2) Results: We illustrate the results of training explanatory performance models for each subject system in Figure 2. Each row shows the distribution of the relative performance influence of a configuration option across the set of tested workloads. For this visualization, we made some tweaks to highlight a few properties: First, we show each regression coefficient as an individual black rug (vertical bar). Second, we highlight the greatest positive influence and smallest minimum influence in red and green, respectively, to illustrate both the range of influences and possible opposing influences (i.e., a performance degrading option becomes performance improving or vice versa).

We have identified three characteristic (but non-exclusive) traits, by which we can describe the distributions of regression coefficients:

① *Spread* of performance influences: Some distributions scatter over a wide range, while others are concentrated around a single value. Consider h2, for which we observe a relative performance difference of 200 % for option MVSTORE, meaning that the performance influence of turning on this option can be twice as high depending on the workload.

② *Opposing influences*: Some distributions exhibit both positive and negative coefficients, while others remain consistent, either positive or negative. For dconvert, we observe both positive and negative influences for option floor for two workloads, while for the remaining workloads, the option has neglible influence.

③ *Conditional influences*: For some models, the majority of coefficients is negligible, but for few workloads we observe options having an influence. For example, for kanzi, we observe for option BWT a negative influence only for a specific workload, whereas for all others the option has no influence.

These criteria, the spread (①) or concentration, opposing influences (②), and options becoming influential only on occasions (③), allow us to group each configuration option into a specific category, as presented in Table V. We omitted all configuration options with low spread and a neglible performance influence since these are not interacting with the workload either.

With two exceptions, we see that all software systems have configuration options, for which, at least, one of the three characteristic traits apply. For jump3r, we found only

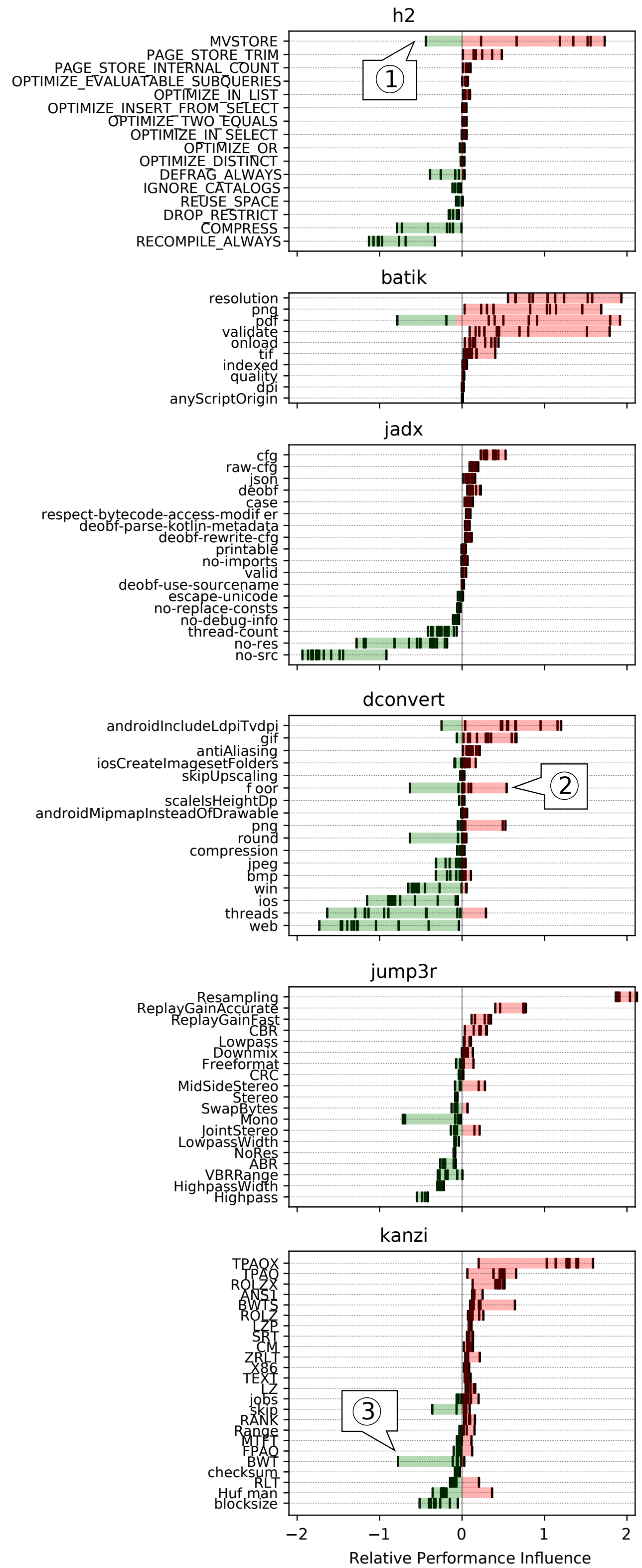


Figure 2: Relative performance influences (*standardized regression coefficients*) for all configuration options across all workloads. Each black bar denotes a workload-specific performance influence. For each configuration option, we highlight the range of observed influences.

Table V: Classification of relative performance influence distributions with respect to *spread*, *opposing influence*, and *conditional influence.*

Category	System	#	Options
① High Spread	kanzi	1	TPAQX
	dconvert	6	androidIncludeLdpiTvdpi, gif, win, ios, threads, web
	h2	3	RECOMPILE_ALWAYS, MVSTORE, COMPRESS
	batik	4	resolution, png, pdf, validate
	jadx	2	no-res, no-src
② Opposing Influences	kanzi	2	Huffman, RLT
	dconvert	3	threads, androidIncludeLdpiTvdpi, floor
	h2	1	MVSTORE
	batik	1	pdf
③ Conditional Influences	jump3r	3	Mono, MidSideStereo, JointStereo
	kanzi	4	BWT, skip, TPAQ, TPAQX
	dconvert	7	floor, png, round, threads, web, ios, win
	h2	2	DEFRAG_ALWAYS, COMPRESS
	batik	3	onload, tiff, png

conditional influences for three options, whereas for jadx, we observed only three options with high spread of relative performance influences. As these differences arise from varying the workload, we conclude that input sensitivity of a configuration option can manifest in different ways, as shown by the three characteristics.

Summary (RQ_2): Configuration options are profoundly input sensitive: We observe high performance variations, non-monotonic behavior, conditional influence, and even diametrically opposed influences for a single option.

C. Code Coverage and Performance (RQ_3)

1) Operationalization: From the findings of RQ_2, we have learned that input sensitivity of configuration options is specific to certain configuration options and can be diverse along multiple dimensions. With regard to the characteristic traits from RQ_2, we conjecture that workload-specific sign flipping or conditional influences are driving factors for non-monotonicity across entire performance distributions (cf. RQ_1). From these findings, the question arises: What causes these different aspects of input sensitivity? For a practical setting, one might, in addition, ask whether it is possible to identify input-sensitive configuration options without the effort of measuring substantial portions of the configuration space under varying worklods. To shed light on possible explanations for input sensitivity, in general, and, possibly, different shades (cf. RQ_2), we switch to the code level and analyze the relation of performance influences inferred for each configuration option to code coverage information.

We augment our performance observations with code coverage information to assess differences in the execution under different workloads. Specifically, we are interested in code sections that implement option-specific functionality (i.e., functionality that is used only if the configuration option is selected). From comparing the coverage information of option-specific code, we can formulate different hypothetical scenarios explaining input sensitivity.

First, if we observe that the coverage of option-specific code is conditioned by the presence of some workload characteristic, we expect that such an option is only influential under respective workloads. This scenario would enables us (to some extent) to use code coverage as a cheap-to-compute proxy for estimating the representativeness of a workload and, by extension, resulting performance models: For options that are known to condition code sections, we can maximize option-code coverage to elicit all option-specific behavior and, thus, performance influence. For instance, a database system could cache a specific view only if a minimum number of queries are executed. Here, the effect of any caching feature would be conditioned by the number of transactions resulting from the workload.

Second, if we observe performance variation across workloads in spite of similar or identical option-specific code coverage, we draw a different picture. Here, we cannot attribute performance variation to code coverage, yet have to consider differences in the workloads' characteristics as potential cause: The presence of a workload characteristic may influence not *what* code sections are executed, but *how* code sections are executed. For instance, in a simple case, a software system's performance may scale linearly with the input size. In a more complex case, the presence of a characteristic may determine how frequently an operation is repeated, as is the case for a database merge. Here, we would not elicit the worst-case performance if a previous transaction has sorted the data (e.g., by building an index).

2) Locating Configuration-Dependent Code: To reason about option-specific code, we require a mapping of configuration options to code. The problem of determining which code section implements which functionality in a software system is known as *feature location* [Rubin and ChechikRubin and Chechik2013]. While there are a number of approaches based on static [Velez, Jamshidi, Sattler, Siegmund, Apel, and KästnerVelez et al.20 [Lillack, Kästner, and BoddenLillack et al.2018], [Luo, Bodden, and SpäthLuo et al.2019] and dynamic taint analysis [Bell and KaiserBell and Kaiser2014], [Velez, Jamshidi, Siegmund, Apel, and KästnerVelez et al.2021], [Kim, Marinov, Khurshid, Batory, Souto, Barros, and D'AmorimKim et a we employ a more lightweight, but also less precise approach that uses code coverage information, such as execution traces. The rationale is that, by exercising feature code, for instance via enabling configuration options or running corresponding tests, its location can be inferred from differences in code coverage. Applications of such an approach have been studied not only for feature location [Wong and LiWong and Li2005], [Sulír and PorubänSulír and Porubän2015], [Michelon, Sotto-Mayor, Martinez, Arrieta, Abreu, and AssunçãoMichelon et al.2021], [Perez and AbreuPerez and Abreu2016], but root work on in program comprehension [Wilde and CaseyWilde and Casey1996], [Wilde and ScullyWilde and Scully1995], [Sherwood and MurphySherwood and Murphy2008], [Perez and AbreuPerez and Abreu2014], [Castro, Perez, and AbreuCastro et al.2019] and fault localization [Agrawal, Horgan, London, and WongAgrawal et al.1995], [Wong, Gao, Li, Abreu, and WotawaWong et al.2016].

Specifically, we follow a strategy akin to *spectrum-based feature location* [Michelon, Sotto-Mayor, Martinez, Arrieta, Abreu, and AssunçãoMichelon et al.2021]: First, we obtain a baseline of *all* option code in the scope of the entire workload selection. For each workload $w \in W$, we compute the set of code lines that depend on any option $o \in O$. Let C_o be the set of configurations with option o selected, and $C_{\neg o}$ with option o deselected. To obtain the code sections specific to option o under workload w, $S_{w,o}$, we subtract the set of the code lines covered under $C_{\neg o}$ from those of C_o:

$$S_{w,o} = \bigcup_{p \in C_o} S_w(p) \setminus \bigcup_{q \in C_{\neg o}} S_w(q) \tag{1}$$

While $S_{w,o}$ yields an approximation of option-dependent code for a single workload, we aggregate the approximations for each workload $w \in W$ to obtain the set of lines that depend on a configuration option o and are executed in, at least, one workload, S_o:

$$S_o = \bigcup_{w \in W} S_{w,o} \tag{2}$$

While this aggregated set is not a ground truth per se, it enables us to reason about differences in option-dependent code in the scope of our selected workloads. That is, the expressiveness of this baseline depends on the diversity of the workloads in question. From the ratio of option-specific code per workload to option-specific code across workloads, $|S_{w_1,o}| / |S_{w_2,o}|$, we can estimate the coverage of option-dependent code.

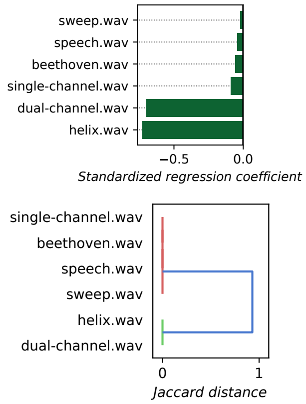
3) Comparing Execution Footprints: From (a) the information about which code sections are specific to a configuration option and (b) how much of these sections is actually covered under different workloads, we can compare the workload-specific execution footprint for each option. By comparing the sets $S_{w_1,o}$ and $S_{w_2,o}$ for any two workloads w_1 and w_2, we can estimate similarity between the option-code coverage via the Jaccard set similarity index. A Jaccard similariy of zero implies that there is no overlap in the lines covered under each workload, whereas a Jaccard similarity of 1 implies that the exact same code was covered. Based on this pairwise similarity metric $sim_o(w_1, w_2)$, we can compute a distance $d_o(w_1,w_2) = 1 - sim(w_1,w_2)$ and cluster all workload-specific execution profiles.

We use agglomerative hierarchical clustering with full linkage to construct dendrograms, as shown in Figure 3. In this bottom-up approach, we iteratively add execution footprints to clusters and merge sub clusters into larger ones depending on their Jaccard similarity to each other. The vertical bars with respect to the x-axis denote the Jaccard distance between merged clusters or, for initial clusters, constituent execution footprints. Finally, we compare (a) the clustering of workload-specific execution profiles for each option with (b) the distribution of relative performance influences for the respective option. As a recap, the distribution of performance influences refers to individual rows in Figure 2. In essence, we ask whether variation in the observed distribution of relative performance influences correspond to similarities or differences in what, or what portions of, option-specific code is executed. Of special interest are the patterns identified in Section IV-B2 and the involved configuration options from Table V.

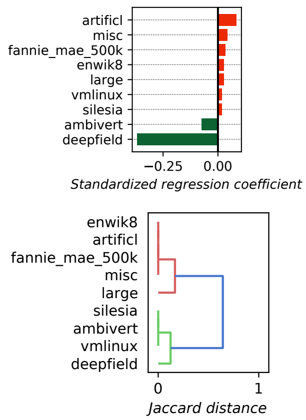
4) Results: We inspected manually the relation of the coverage similarity across workloads per option and the observed workload-specific performance influences. For the majority of cases, the results suggest that the workload determines how the code is executed since we did not observe a strong relationship between performance variation and differences in code coverage. However, we have identified seven configuration options across three software systems (jump3r, kanzi, and dconvert) for which code coverage is likely the driving factor for performance variation. We present illustrative examples for both scenarios in Figure 3.

For the scenario of a workload conditioning code coverage, jump3r is a good illustrative example in Figure 3a: We have identified two workloads that exhibit multiple audio channels (helix.wav and dual-channel.wav) under which configuration options (Mono, MidSideStereo and JointStereo) become influential. This is supported by the coverage information we collected, where we see that both workloads result in similar code sections covered, whereas such code sections are not covered under other workloads. We observe similar effects for kanzi and dconvert. For kanzi, we find two distinct clusters of code coverage whose workloads show opposing influences for option skip (cf. Figure 3b). For dconvert, under workload svg-large no option-specific code is executed, resulting in little influence for options web, ios, and gif (cf. Figure 3c). These three examples suggest that a workload can indeed determine whether a configuration option's code section is executed and thus determine whether this option is influential and whether its performance influence is positive or negative.

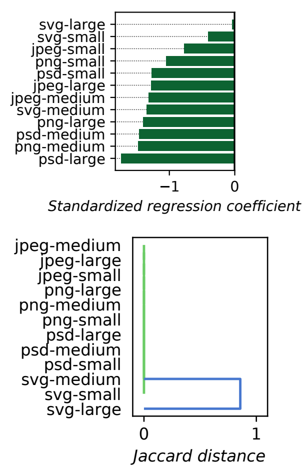
By contrast, the majority of cases we inspected did not show such a relationship. We illustrate one example that



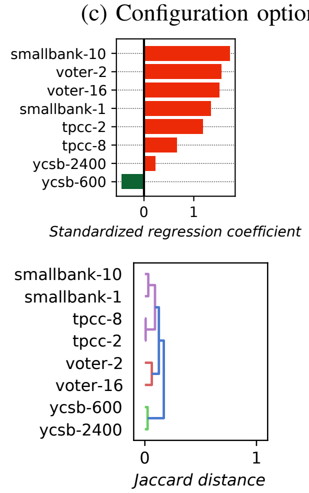
(a) Configuration option ≪Mono≫ of jump3r



(b) Configuration option ≪skip≫ of kanzi



(c) Configuration option ≪web≫ of dconvert



(d) Configuration option ≪COMPRESS≫ of h2

highlights that non-linear shifts in performance influence can arise even from little differences in the workload. For h2, we pursued a rather controlled experiment. Here, we vary the scale factor for four different standard benchmarks. Thus, we can expect some inherent similarity across these pairs. In Figure 3d, we show the comparison of performance influences and the corresponding dendrogram for the configuration option COMPRESS. While we see that the execution footprints for the pairs of benchmarks form clusters and are quite similar (i.e., the distance is below 0.3 among all workloads), we still observe considerable performance variation, most notably, in the case of workload ycsb-600. So, contrary to our expectation, varying only one characteristic (here, the scale factor) can introduce further variation that cannot be plausibly explained by workload-specific code coverage.

For the other subject systems and remaining configuration options, we could not find any relation between workload-dependent performance variations and code coverage. We found both cases, differences in code coverage do not correspond to variation in the relative performance influences and vice versa. That is, for the majority of configuration options, input sensitivity cannot be explained by a varying option-specific code coverage. Instead, workload characteristics most likely account for variation in how covered option specific-code is executed, including the loop passes and method calls as well as variation arising from method arguments.

Summary (RQ_3): Varying the workload can condition the execution of option-specific code (*code coverage*) and cause performance differences. However, there is no single driving factor: *Code utilization* depending on workload characteristics is a likely factor accounting for the majority of performance influence variation.

V. Discussion / Lessons Learned

Our experiments shed light on the effects of varying workloads on configuration-specific performance as well as which code sections are executed. We now relate our findings to the strategies and challenges outlined in Section 2, mainly strategies for considering the workload when modeling performance as well as the challenges with representative workloads. The remainder of this section addresses threats to validity.

A. Incorporating Workloads

Existing strategies to consider the workload when modeling performance either include workload characteristics (scale factor, size, etc.) as independent variables alongside configuration options [Koc, Mordahl, Wei, Foster, and PorterKoc et al.2021] or transfer existing models (learned from a single workload) to another workload. While the first strategy appears straightforward, characterization of the workload is highly domain-specific and requires in-depth understanding of operation measured, which is more suited to analytical performance models [Brown, Falgout, Jones, Jim, and JonesBrown et al.2000], [Gahvari, Baker, Schulz, Yang, Jordan, and GroppGahvari et al.2011]. By contrast, the former strategy can—in theory—be facilitated with only an existing model learned from a single workload and relatively few configuration-specific measurements under a second workload [Martin, Acher, Lesoil, Jezequel, Khelladi, and PereiraMartin et al.2021].

1) *Single-transfer Scenario:*

2) *Multi-transfer Scenario:* We do...

B. Workload Composition

The creation of a representative workload is a challenge for effective testing for both functional and non-functional properties. While we cannot guarantee that the combination of workloads in our study itself triggers all functionality, we have identified scenarios, where option-specific code coverage corresponds with changes in the respective option's influence on performance [cf ...].

VI. Threats to Validity

Threats to *internal validity* include measurement noise which may distort our classification into categories (Section IV-A) and model construction (Section IV-B). We mitigate these threats by repeating each experiment five times and reporting the median as a robust measure in a controlled environment. Moreover, the coverage analysis (cf. Section ?) entails a noticeable instrumentation overhead, which may distort performance observations. We mitigate this threat by separating the experiment runs for coverage assessment and performance measurement. In the case of h2, the load generator of the OLTPBench framework [Difallah, Pavlo, Curino, and Cudre-MaurouxDifallah et al.201 ran on the same machine as the database since we were testing an embedded scenario with only negligible overhead.

Threats to *external validity* include the selection of subject systems and workloads. To ensure generalizability, we select software systems from various various application domains as well as two different programming language ecosystems (cf. Table I). In lieu of domain knowledge, we cannot select workloads systematically with respect to workload characteristics due to workloads inherent opacity. We address this issue by varying likely relevant characteristics and, where possible, reusing workloads across subject systems of the same domain. Achieving true representativeness is desirable, yet intractable. The goal of this selection is to study the *presence* and quality of workload-option interactions, but not their *prevalence*. Hence, we believe this selection does not invalidate our findings.

VII. Conclusion

Most modern software systems exhibit configuration options to customize behaviors and meet user demands. Configuration choices, however, can also affect the performance of a software system. State-of-the-art approaches model configuration-dependent software performance, yet overlook variation due to the workload. Until now, there exists no *systematic* assessment of what is driving the effect that input sensitivity of individual configuration options' influence on performance. We have conducted an empirical study of 29 347 configurations and 55 workloads across six configurable software systems to characterize the effects that varying the workload can have on configuration-specific performance. We compare performance measurements with code coverage data to identify possible factors that drive input sensitivity. We found that the interactions between options and workloads are driven by workload characteristics conditioning the execution of option-specific code sections as well as determining how option-specific code sections are executed. Our findings highlight the necessity to consider input sensitivity when modeling configuration-dependent performance as varying the workload resulted in a substantial number of non-monotonous relationships, which limits a performance model's representativeness. Code analysis can provide an effective strategy to differentiate between sources of input sensitivity to obtain more representative performance models.

References

[Agrawal, Horgan, London, and WongAgrawal et al.1995] Hiralal Agrawal, Joseph R. Horgan, Saul London, and W. Eric Wong. 1995. Fault localization using execution slices and dataflow tests. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*. 143–151. https://doi.org/10.1109/ISSRE.1995.497652

[Bao, Busany, Lo, and MaozBao et al.2019] Lingfeng Bao, Nimrod Busany, David Lo, and Shahar Maoz. 2019. Statistical Log Differencing. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 851–862. https://doi.org/10.1109/ASE.2019.00084

[Bell and KaiserBell and Kaiser2014] Jonathan Bell and Gail Kaiser. 2014. Phosphor: Illuminating Dynamic Data Flow in Commodity Jvms. *ACM SIGPLAN Notices* 49, 10 (Oct. 2014), 83–101. https://doi.org/10.1145/2714064.2660212

[Brown, Falgout, Jones, Jim, and JonesBrown et al.2000] Peter N. Brown, Robert D. Falgout, Jim E. Jones, Jim, and E. Jones. 2000. Semicoarsening Multigrid On Distributed Memory Machines. *SIAM Journal on Scientific Computing* 21 (2000), 1823–1834. https://doi.org/10.1137/S1064827598339141

[Calzarossa, Massari, and TesseraCalzarossa et al.2016] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. 2016. Workload Characterization: A Survey Revisited. *ACM Computer Survey* 48, 3 (Feb. 2016). https://doi.org/10.1145/2856127

[Castro, Perez, and AbreuCastro et al.2019] Bruno Castro, Alexandre Perez, and Rui Abreu. 2019. Pangolin: An SFL-Based Toolset for Feature Localization. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 1130–1133. https://doi.org/10.1109/ASE.2019.00119

[Ceesay, Lin, and BarkerCeesay et al.2020] Sheriffo Ceesay, Yuhui Lin, and Adam Barker. 2020. A Survey: Benchmarking and Performance Modelling of Data Intensive Applications. In *IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*. 67–76. https://doi.org/10.1109/BDCAT50828.2020.00012

[CliffCliff1993] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114 (1993), 494–509.

[Curtsinger and BergerCurtsinger and Berger2013] Charlie Curtsinger and Emery D. Berger. 2013. STABILIZER: Statistically Sound Performance Evaluation. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. ACM, 219–228. https://doi.org/10.1145/2451116.2451141

[DaoudDaoud2017] Jamal I. Daoud. 2017. Multicollinearity and Regression Analysis. *Journal of Physics: Conference Series* 949 (dec 2017), 012009. https://doi.org/10.1088/1742-6596/949/1/012009

[Difallah, Pavlo, Curino, and Cudre-MaurouxDifallah et al.2013] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings of the VLDB Endowment* 7, 4 (Dec. 2013), 277–288. https://doi.org/10.14778/2732240.2732246

[Ding, Ansel, Veeramachaneni, Shen, O'Reilly, and AmarasingheDing et al.2015] Yufei Ding, Jason Ansel, Kalyan Veeramachaneni, Xipeng Shen, Una-May O'Reilly, and Saman Amarasinghe. 2015. Autotuning Algorithmic Choice for Input Sensitivity. *SIGPLAN Not.* 50, 6 (jun 2015), 379–390. https://doi.org/10.1145/2813885.2737969

[Ding, Pervaiz, Krishnan, and HoffmannDing et al.2020] Yi Ding, Ahsan Pervaiz, Sanjay Krishnan, and Henry Hoffmann. 2020. *Bayesian Learning for Hardware and Software Configuration Co-Optimization*. Technical Report 13. University of Chicago.

[Dorn, Apel, and SiegmundDorn et al.2020] Johannes Dorn, Sven Apel, and Norbert Siegmund. 2020. Mastering Uncertainty in Performance Estimations of Configurable Software Systems. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. ACM, 684–696. https://doi.org/10.1145/3324884.3416620

[Falkner, Lindauer, and HutterFalkner et al.2015] Stefan Falkner, Marius Lindauer, and Frank Hutter. 2015. SpySMAC: Automated Configuration and Performance Analysis of SAT Solvers. In *Theory and Applications of Satisfiability Testing – SAT 2015*, Marijn Heule and Sean Weaver (Eds.). Springer International Publishing, 215–222.

[Gahvari, Baker, Schulz, Yang, Jordan, and GroppGahvari et al.2011] Hormozd Gahvari, Allison H. Baker, Martin Schulz, Ulrike Meier Yang, Kirk E. Jordan, and William Gropp. 2011. Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms. In *Proceedings of the International Conference on Supercomputing (ICS '11)*. ACM, New York, NY, USA, 172–181. https://doi.org/10.1145/1995896.1995924

[Guo, Czarnecki, Apel, Siegmund, and WasowskiGuo et al.2013] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware Performance Prediction: A Statistical Learning Approach. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 301–311. https://doi.org/10.1109/ASE.2013.6693089

[Guo, Yang, Siegmund, Apel, Sarkar, Valov, Czarnecki, Wasowski, and YuGuo et al.2018] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2018. Data-efficient performance learning for configurable systems. *Empirical Software Engineering* 23, 3 (2018), 1826–1867. https://doi.org/10.1007/s10664-017-9573-6

[Ha and ZhangHa and Zhang2019a] Huong Ha and Hongyu Zhang. 2019a. DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 1095–1106. https://doi.org/10.1109/ICSE.2019.00113

[Ha and ZhangHa and Zhang2019b] Huong Ha and Hongyu Zhang. 2019b. Performance-Influence Model for Highly Configurable Software with Fourier Learning and Lasso Regression. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. 470–480. https://doi.org/10.1109/ICSME.2019.00080

[Henard, Papadakis, Harman, and Le TraonHenard et al.2015] 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 *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 517–528. https://doi.org/10.1109/icse.2015.69

[Jamshidi, Siegmund, Velez, Kästner, Patel, and AgarwalJamshidi et al.2017a] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017a. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 497–508. https://doi.org/10.1109/ASE.2017.8115661

[Jamshidi, Velez, Kästner, and SiegmundJamshidi et al.2018] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 71–82. https://doi.org/10.1145/3236024.3236074

[Jamshidi, Velez, Kästner, Siegmund, and KawthekarJamshidi et al.2017b] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017b. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 31–41. https://doi.org/10.1109/SEAMS.2017.11

[Jiang and HassanJiang and Hassan2015] Zhen Ming Jiang and Ahmed E. Hassan. 2015. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Transactions on Software Engineering (TSE)* 41, 11 (2015), 1091–1118. https://doi.org/10.1109/TSE.2015.2445340

[Kaltenecker, Grebhahn, Siegmund, and ApelKaltenecker et al.2020] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. 2020. The Interplay of Sampling and Machine Learning for Software Performance Prediction. *IEEE Software* PP (04 2020). https://doi.org/10.1109/MS.2020.2987024

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

[KendallKendall1938] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[Khavari Tavana, Sun, Bohm Agostini, and KaeliKhavari Tavana et al.2019] Mohammad Khavari Tavana, Yifan Sun, Nicolas Bohm Agostini, and David Kaeli. 2019. Exploiting Adaptive Data Compression to Improve Performance and Energy-Efficiency of Compute Workloads in Multi-GPU Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 664–674. https://doi.org/10.1109/IPDPS.2019.00075

[Kim, Marinov, Khurshid, Batory, Souto, Barros, and D'AmorimKim et al.2013] Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don Batory, Sabrina Souto, Paulo Barros, and Marcelo D'Amorim. 2013. SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 257–267. https://doi.org/10.1145/2491411.2491459

[Koc, Mordahl, Wei, Foster, and PorterKoc et al.2021] Ugur Koc, Austin Mordahl, Shiyi Wei, Jeffrey S Foster, and Adam A Porter. 2021. SATune: A Study-Driven Auto-Tuning Approach for Configurable Software Verification Tools. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 330–342. https://doi.org/10.1109/ASE51524.2021.9678761

[Kounev, Lange, and von KistowskiKounev et al.2020] Samuel Kounev, Klaus-Dieter Lange, and Jóakim von Kistowski. 2020. *Systems Benchmarking* (1 ed.). Springer International Publishing. https://doi.org/10.1007/978-3-030-41705-5

[Liao, Chen, Li, Zeng, Shang, Guo, Sporea, Toma, and SajediLiao et al.2020] Lizhi Liao, Jinfu Chen, Heng Li, Yi Zeng, Weiyi Shang, Jianmei Guo, Catalin Sporea, Andrei Toma, and Sarah Sajedi. 2020. Using black-box performance models to detect performance regressions under varying workloads: an empirical study. *Empirical Software Engineering (ESE)* (2020), 1–31. https://doi.org/10.1007/s10664-020-09866-z

[Lillack, Kästner, and BoddenLillack et al.2018] Max Lillack, Christian Kästner, and Eric Bodden. 2018. Tracking Load-Time Configuration Options. *IEEE Transactions on Software Engineering (TSE)* 44, 12 (2018), 1269–1291. https://doi.org/10.1109/TSE.2017.2756048

[LovricLovric2010] Miodrag Lovric. 2010. *International Encyclopedia of Statistical Science* (1st edition. ed.). Springer.

[Luo, Bodden, and SpäthLuo et al.2019] Linghui Luo, Eric Bodden, and Johannes Späth. 2019. A Qualitative Analysis of Android Taint-Analysis Results. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 102–114. https://doi.org/10.1109/ASE.2019.00020

[Maricq, Duplyakin, Jimenez, Maltzahn, Stutsman, and RicciMaricq et al.2018] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming performance variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 409–425.

[Martin, Acher, Lesoil, Jezequel, Khelladi, and PereiraMartin et al.2021] Hugo Martin, Mathieu Acher, Luc Lesoil, Jean Marc Jezequel, Djamel Eddine Khelladi, and Juliana Alves Pereira. 2021. Transfer Learning Across Variants and Versions: The Case of Linux Kernel Size. *IEEE Transactions on Software Engineering (TSE)* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3116768

[Maxiaguine, Liu, Chakraborty, and OoiMaxiaguine et al.2004] A. Maxiaguine, Yanhong Liu, S. Chakraborty, and Wei Tsang Ooi. 2004. Identifying "representative" workloads in designing MpSoC platforms for media processing. In *2nd Workshop o nEmbedded Systems for Real-Time Multimedia, 2004. ESTImedia 2004*. 41–46. https://doi.org/10.1109/ESTMED.2004.1359702

[Michelon, Sotto-Mayor, Martinez, Arrieta, Abreu, and AssunçãoMichelon et al.2021] Gabriela K. Michelon, Bruno Sotto-Mayor, Jabier Martinez, Aitor Arrieta, Rui Abreu, and Wesley KG Assunção. 2021. Spectrum-based feature localization: a case study using ArgoUML. In *Proceedings of the International Conference on Software Product Lines (SPLC)*. ACM, 126–130. https://doi.org/10.1145/3461001.3473065

[Nair, Menzies, Siegmund, and ApelNair et al.2017] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using Bad Learners to Find Good Configurations. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 257–267. https://doi.org/10.1145/3106237.3106238

[Nair, Yu, Menzies, Siegmund, and ApelNair et al.2020] Vivek Nair, Zhe Yu, Tim. Menzies, Norbert Siegmund, and Sven Apel. 2020. Finding Faster Configurations Using FLASH. *IEEE Transactions on Software Engineering (TSE)* 46, 7 (2020), 794–811. https://doi.org/10.1109/TSE.2018.2870895

[O'BrienO'Brien2007] Robert M. O'Brien. 2007. A caution regarding rules of thumb for variance inflation factors. *Quality & Quantity* 41, 5 (2007), 673–690.

[Oh, Batory, Myers, and SiegmundOh et al.2017] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-Optimal Configurations in Product Lines by Random Sampling. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 61–71. https://doi.org/10.1145/3106237.3106273

[Papadopoulos, Versluis, Bauer, Herbst, Kistowski, Ali-Eldin, Abad, Amaral, Tůma, and Ios Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim von Kistowski, Ahmed Ali-Eldin, Cristina L. Abad, José Nelson Amaral, Petr Tůma, and Alexandru Iosup. 2021. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering (TSE)* 47, 8 (2021), 1528–1543. https://doi.org/10.1109/TSE.2019.2927908

[Pereira, Acher, Martin, and JézéquelPereira et al.2020] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, 277–288. https://doi.org/10.1145/3358960.3379137

[Perez and AbreuPerez and Abreu2014] Alexandre Perez and Rui Abreu. 2014. A Diagnosis-Based Approach to Software Comprehension. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. ACM, 37–47. https://doi.org/10.1145/2597008.2597151

[Perez and AbreuPerez and Abreu2016] Alexandre Perez and Rui Abreu. 2016. Framing program comprehension as fault localization. *Journal of Software: Evolution and Process* 28 (2016), 840–862. https://doi.org/10.1002/smr.1799

[Plotnikov, Melnik, Vardanyan, Buchatskiy, Zhuykov, and LeePlotnikov et al.2013] Dmitry Plotnikov, Dmitry Melnik, Mamikon Vardanyan, Ruben Buchatskiy, Roman Zhuykov, and Je-Hyung Lee. 2013. Automatic Tuning of Compiler Optimizations and Analysis of their Impact. *Procedia Computer Science* 18 (2013), 1312–1321. https://doi.org/10.1016/j.procs.2013.05.298 2013 International Conference on Computational Science.

[Romano, Kromrey, Coraggio, Skowronek, and DevineRomano et al.2006] Jeanine Romano, Jeffrey D. Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices?. In *Annual Meeting of the Southern Association for Institutional Research*. 1–51.

[Rubin and ChechikRubin and Chechik2013] Julia Rubin and Marsha Chechik. 2013. A Survey of Feature Location Techniques. In *Domain Engineering: Product Lines, Languages, and Conceptual Models*, Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, and Jorn Bettin (Eds.). Springer, 29–58. https://doi.org/10.1007/978-3-642-36654-3_2

[Sarkar, Guo, Siegmund, Apel, and CzarneckiSarkar et al.2015] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, 342–352. https://doi.org/10.1109/ASE.2015.45

[Sherwood and MurphySherwood and Murphy2008] Kaitlin Duck Sherwood and Gail C. Murphy. 2008. *Reducing Code Navigation Effort with Differential Code Coverage*. Technical Report 14. Department of Computer Science, University of British Columbia.

[Shu, Sui, Zhang, and XuShu et al.2020] Yangyang Shu, Yulei Sui, Hongyu Zhang, and Guandong Xu. 2020. Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Article 16, 11 pages. https://doi.org/10.1145/3382494.3410677

[Siegmund, Grebhahn, Apel, and KästnerSiegmund et al.2015] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-Influence Models for Highly Configurable Systems. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 284–294. https://doi.org/10.1145/2786805.2786845

[Siegmund, Kolesnikov, Kästner, Apel, Batory, Rosenmüller, and SaakeSiegmund et al.2012 Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting

Performance via Automated Feature-Interaction Detection. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 167–177. https://doi.org/10.1109/ICSE.2012.6227196

[Sulír and PorubänSulír and Porubän2015] Matúš Sulír and Jaroslav Porubän. 2015. Semi-automatic concern annotation using differential code coverage. In *Proceedings of the IEEE International Scientific Conference on Informatics (ISCI)*. 258–262. https://doi.org/10.1109/Informatics.2015.7377843

[Valov, Petkovich, Guo, Fischmeister, and CzarneckiValov et al.2017] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *ICPE*. ACM, 39–50. https://doi.org/10.1145/3030207.3030216

[Velez, Jamshidi, Sattler, Siegmund, Apel, and KästnerVelez et al.2020] Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, and Christian Kästner. 2020. ConfigCrusher: Towards White-Box Performance Analysis for Configurable Systems. *Automated Software Engineering (ASE)* (2020), 1–36. https://doi.org/10.1007/s10515-020-00273-8

[Velez, Jamshidi, Siegmund, Apel, and KästnerVelez et al.2021] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. 2021. White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 1072–1084. https://doi.org/10.1109/ICSE43902.2021.00100

[Weber, Apel, and SiegmundWeber et al.2021] Max Weber, Sven Apel, and Norbert Siegmund. 2021. White-Box Performance-Influence Models: A Profiling and Learning Approach. *Proceedings of the International Conference on Software Engineering (ICSE)*, 1059–1071. https://doi.org/10.1109/ICSE43902.2021.00099

[Wilde and CaseyWilde and Casey1996] Norman Wilde and Christopher Casey. 1996. Early Field Experience with the Software Reconnaissance Technique for Program Comprehension. In *Proceedings of the International Conference on Software Maintenance (ICSM)*. 312–318. https://doi.org/10.1109/ICSM.1996.565034

[Wilde and ScullyWilde and Scully1995] Norman Wilde and Michael C. Scully. 1995. Software reconnaissance: Mapping program features to code. *Journal of Software Maintenance: Research and Practice* 7, 1 (1995), 49–62. https://doi.org/10.1002/smr.4360070105

[Wong, Gao, Li, Abreu, and WotawaWong et al.2016] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering (TSE)* 42, 8 (2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368

[Wong and LiWong and Li2005] W E Wong and J Li. 2005. An integrated solution for testing and analyzing Java applications in an industrial setting. In *Proceedings of the Asia-Pacific Conference on Software Engineering (ASPEC)*. 8. https://doi.org/10.1109/APSEC.2005.39

[Xu, Hutter, Hoos, and Leyton-BrownXu et al.2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2008. SATzilla: Portfolio-Based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research* 32, 1 (jun 2008), 565–606.

[Zhang, Guo, Blais, and CzarneckiZhang et al.2015] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. 2015. Performance Prediction of Configurable Software Systems by Fourier Learning (T). In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 365–373. https://doi.org/10.1109/ASE.2015.15