Chapter 1

Introduction

Nowadays, most software systems provide configuration options that allow end-users, developers, and administrators to adapt them to their specific needs. But this adaptability comes at a high cost.
The more options there are to configure a system, the harder it gets to find configurations optimal for the specific needs of the user. While the functional aspects of a configuration option are usually well known and documented, the non-functional properties are mostly unknown. This makes it difficult to configure a software system optimal for a non-functional property such as performance, power consumption or memory footprint.
The naive way to test all possible configurations in order to determine the optimal configuration for a given non-functional property is unfeasible. This is due to the combinatorial complexity of the configuration space and the constraints on the configuration options. Since there is no straightforward way to determine the influence of configuration options on non-functional properties, these properties are mostly ignored in the choice of a fitting configuration [**?**] or are only chosen by previous "tribal knowledge", possibly leaving already built-in optimizations in software systems untapped.

In literature, a commonly used method to help choosing the optimal configuration for a non-functional property is to create a model describing the influence of configuration options on the non-functional property [**?**]. This model can then be used to find optimal configurations and predict the non-functional property of previously unseen configurations without any further testing of the system. A developer, for example, might use this model to find a configuration that minimizes the power consumption of the software to reduce operational costs. To create this kind of model, machine learning techniques are a simple go-to. The creation of such models is a research field of its own and there are many well-understood machine-learning techniques, which can be used to create a model for the influence of software configurations

on non-functional properties. Since it is infeasible to test all possible configurations, a subset of configurations must be selected. This process is called sampling. While sampling in traditional machine learning applications might be as easy as just randomly picking data points to train the model, it is not as straightforward for configurations options in the presence of constraints. This is mainly due to the constraints the configuration options have, which make the sampling of configurations into a task of finding a valid assignment of configuration options and thus NP-hard.

There are several techniques already tackling the problem of sampling on software configurations. These techniques allow a user to sample software configurations and select random or near-random configurations under specific criteria. With these sampling techniques, the creation of performance influence models with prediction errors of 1% are possible [**?**]. However, there is still motivation to further reduce the needed amount of samples to make predictions more performant and easier to use. This reduction of required data to model the influence of multiple parameters on a specific variable is not a problem specific to software configurations. Work by **?**] and **?**] presents an experimental design for sensitivity analysis which reduces the number of needed simulations to less than the number of parameters and which is still able to give meaningful information of the influence of those parameters. This experimental design is called group sampling. While the core concept of analyzing the influence of multiple parameters on a single variable is quite similar to the problem of analyzing the influence of configuration options on non-functional properties, there are a few problems in adapting these techniques to software configurations. As already mentioned before, this is mainly due to the constraints and interactions of configuration options.

The constraints on configuration options make it hard to look at the influence of a single configuration option. This is because one option might not be able to be turned on independently of other configuration options. Therefore, measurements conducted on this pair of configuration options might not be able to tell the specific influence of the single configuration options in question. Further, certain configuration options might interact with each other, cancelling out or multiplying their influences. This is easily understood by looking at two common configuration options in compression programs. The encryption and the compression. If the compression is more aggressively compressing the data, the encryption part of the program has fewer data to encrypt, making the program more performant.

In this work, we analyze how the experimental design of group sampling can be adapted to software configurations, by implementing a group sampling algorithm for software configurations and answering the following questions:

RQ1 How can we mitigate the effect of constraints on the configuration options for a group sampling approach?

In literature, in group sampling, constraints on the parameters are non-existent, but with configuration options, these constraints make it impossible to create arbitrary groupings. We devise two approaches to create groups in the presence of constraints and test which results in a better performance influence model.

RQ2 How effective is a group sampling algorithm compared to random sampling in order to create performance influence models?

We answer this question with experiments on real-world and synthetic data sets. By comparing group sampling and random sampling with linear regression with varying sample and group sizes, we determine under which conditions, if any, group sampling is superior to random sampling.

RQ3 How scalable is group sampling?

Since group sampling is an experimental design for large amounts of parameters, the resulting model should be useful for large software systems where random sampling might not be of use because of the large sample sets required to create a performant model. This is done by creating synthetic data for a software system with a large configuration space.

RQ4 Can group sampling be used to effectively identify feature interactions and include them in the resulting model?

Since the grouping of configuration options already includes the influences of feature interactions, group sampling may be used to identify these interactions and effectively incorporate those in the resulting model. We answer this question with experiments on synthetic data, where the amount of feature interactions is at our discretion.

Further, we will have a look at optimization techniques, which are already described in literature to get better results when using a group sampling algorithm.

First, we give some background knowledge in **??** and then explain our approach to group sampling in Chapter 3. In Chapter 4 and Chapter 5 we evaluate and discuss our approach.