CoMSA: A Modeling-Driven Sampling Approach for Configuration Performance Testing

Yuanjie Xia, Zishuo Ding*, and Weiyi Shang
Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada
Email: {x_yuanji, zi_ding, shang}@encs.concordia.ca

***Abstract*—Highly configurable systems enable customers to flexibly configure the systems in diverse deployment environments. The flexibility of configurations also poses challenges for performance testing. On one hand, there exist a massive number of possible configurations; while on the other hand, the time and resources are limited for performance testing, which is already a costly process during software development. Modeling the performance of configurations is one of the solutions to reduce the cost of configuration performance testing. Although prior research proposes various modeling and sampling techniques to build configuration performance models, the sampling approaches used in the model typically do not consider the accuracy of the performance models, leading to potential sub-optimal performance modeling results in practice. In this paper, we present a modeling-driven sampling approach (CoMSA) to improve the performance modeling of highly configurable systems. The intuition of CoMSA is to select samples based on their uncertainties to the performance models. In other words, the configurations that have the more uncertain performance prediction results by the performance models are more likely to be selected as further training samples to improve the model. CoMSA is designed by considering both scenarios where 1) the software projects do not have historical performance testing results (cold start) and 2) there exist historical performance testing results (warm start). We evaluate the performance of our approach in four subjects, namely LRZIP, LLVM, x264, and SQLite. Through the evaluation result, we can conclude that our sampling approaches could highly enhance the accuracy of the prediction models and the efficiency of configuration performance testing compared to other baseline sampling approaches.**

I. Introduction

Modern software is often highly configurable, which provides flexible running options for customers [34], [38]. Examples of the available configurations of the software include the choices of the algorithm, the value of parameters as thresholds, and the deployment number of the threads. The complexity of the configurations has become ever more challenging for developers to deliver high-quality software systems [19], [46]–[48]. In particular, the choice of configurations can have a high impact on the performance of the software, such as its processing time and memory usage, which may directly affect the experience of end users [1], [2].

Testing the performance of software with different configurations is crucial for highly configurable software [11], [14], [39]. However, large software systems may contain thousands (or even higher) of different combinations of configuration options. A naive approach to conducting performance testing with combinations of configuration options is not just simply time-consuming, but rather impossible. Moreover, as modern software evolves rapidly, testing all the configurations in the software in each update is even more unrealistic.

In order to ensure the performance of highly configurable systems in an efficient matter, one may select a sample of configurations for performance testing. The performance testing results from the samples are used to build a performance model, which can be later used to predict the performance of the configurations that are not tested. Hence, numerous approaches have been developed to construct configuration performance models [8], [11], [13], [14], [25], [39]–[41] such as linear regression, and random forest, which are based on traditional machine learning models. Recently, the use of deep learning models for configuration performance modeling, e.g., DeepPerf [14] and Perf-AL [39], has also seen a significant increase [40], [41].

One of the common challenges of building configuration performance models is how to select samples of configurations as training data. Besides widely using random sampling in the training process of performance models in prior research [11], [13]–[15], [18], [23], [31], [32], [39], prior research has proposed various sampling techniques for configurations. For example, Kaltenecker et al. [20] developed a diversified distance-based sampling approach, which aims to increase the diversity of samples. Meanwhile, numerous sampling approaches [12], [24], [45] aim to cover a broader range of configuration categories. However, none of the existing sampling approaches are driven by the modeling of performance. In other words, the prediction accuracy of the performance models that are built from the samples is not in consideration of the sampling approaches. Such a discrepancy highlights the potential risk where the performance models that are built from the samples are sub-optimal to provide an accurate prediction of the performance of configurations that are not tested.

In this paper, we propose a modeling-driven sampling approach named CoMSA based on the uncertainty of their predictions in performance models. In particular, we design the approach in two scenarios:

- **Cold-start scenario.** Not all software projects maintain historical performance testing results. As a result, we

* Corresponding author.

designed a cold-start approach for selecting samples of configurations without any historical data. Our cold-start approach consists of four parts, which are selector initialization, uncertainty measurement, selector update, and performance model training. We first randomly select small sets of configurations for initializing the selector. Then, the selector measures the uncertainty of samples untested and selects the most uncertain samples of configurations for testing. After repeatedly updating the selector, we use all the tested samples for training a performance model.

- **Warm-start scenario.** During software development, the historical measurements (i.e., testing results) from previous performance testings consist of valuable knowledge of the software performance [18], [23], [28]. Therefore, we leverage the historical measurements to initialize selectors. Furthermore, the training data are augmented with historical measurements.

We evaluate our approaches with four open-source projects, namely LRZIP, LLVM, x264 and SQLite, which are highly configurable and contain development history in their version control repository.

Before conducting the evaluation, we first conduct a preliminary study on the selected uncertain configurations of different performance models (cf. PQ1) and their modeling accuracy (cf. PQ2). We find that the uncertainty of configurations is often related to the performance models. Therefore, the choice of performance models would be important in our evaluation. The study results further show that XGBoost produces the best performance model, hence is later used in our evaluation.

Building on the insights from our preliminary study, we evaluate our approach CoMSA in both cold-start and warm-start scenarios, respectively:

- **Cold-start scenario.** We compare our approach with other sampling approaches in prior researches [6], [16], [27], [31], [40], [45] and a strong random sampling baseline [33] based on the fitness of the performance models that are built from the samples and the prediction accuracy for the configurations that are not tested.
- **Warm-start scenario.** We evaluate our approach by using performance testing results from 1) only one past commit and 2) all past commits in order to assess the value of utilizing historical measurements on configuration sampling and model training.

The results show that our sampling approach produces better performance models than sampling approaches from prior research, in terms of both model fitness and prediction accuracy. Furthermore, the historical measurements can considerably improve the efficiency of configuration performance testing during software development.

Paper organization. The remainder of the paper is organized as follows. Section II introduces the background and related work of our paper. Section III presents our approach in both cold-start and warm-start scenarios. Section IV presents our evaluation settings. Section V presents the results of our preliminary study. Section VI evaluates our approach. Section VII discuss the threats that may influence the validity of our research. Finally, Section VIII concludes the paper.

II. Background and Related Work

In this section, we present the background and related prior research of this paper. Prior studies of highly configurable systems demonstrate the challenge of choosing values from a large number of configurations and the performance variance among the choices of these configurations [20], [30], [34], [48]. Building performance models for these highly configurable systems is one of the common approaches of addressing such challenges [11], [13], [14], [34], [39]. Therefore, in the rest of this section, we first present the purposes of building configuration performance models. Then we present the sampling approaches that are used in prior research to build performance models. Afterwards, we discuss the relationship between the sampling approaches and the purposes in order to finally motivate our approach. We summarize the modeling methods, sampling approaches and the purposes from prior research in Table I.

A. The purposes of configuration performance modeling

In this sub-section, we present the purpose of the configuration performance modeling.

Finding optimal configurations. Some researchers have directed their effort towards finding the optimal deployment configurations for a system [31], [32], while others have focused on identifying the optimal configuration for a specific environment [4], [26], [29]. These approaches may employ statistical recursive searching algorithm [32] or fast sequential model-based method [31].

Test reduction. A common purpose of configuration performance modeling is to reduce the number of configurations that need to be tested. Different modeling approaches, such as decision tree [11], [13] and deep learning models [14], [39], [44] have been leveraged to improve the accuracy of the prediction, In addition, research also focuses on improving the prediction model by enhancing sampling techniques [20], [45]. The purpose of this paper is also to reduce the cost of configuration performance testing by proposing a modeling-driven sampling approach.

Improving interpretability. Prior research develops approaches to reveal the relationship between configurations and to interpret performance testing results. For example, Fourier learning [15] and feature's influence [40], [41] are used to improve the interpretability of performance models. On the other hand, in order to enhance the interpretability of performance models, researchers often use simple regression techniques like linear regressions. However, this may come at the cost of decreased prediction accuracy.

Performance model reuse. Ensuring the reusability of performance models across different versions and environments is crucial in the field of configuration performance prediction. To enhance reusability, transfer learning techniques have been widely applied in performance modeling [18], [23].

TABLE I: The summary of the modeling methods and sampling approaches

Reference (Name)	Modeling method	Sampling approach	Purpose
Guo et al. [11](CART)	Classification and Regression Trees (CART)	Feature-size heuristic	Test reduction
Guo et al. [13] (DECART)	Classification and Regression Trees (CART)	Feature-size heuristic	Test reduction
Ha et al. [15]	Fourier learning and Lasso regression	Random	Improving interpretability
Ha et al. [14] (DeepPerf)	Feedforward Neural Networks(FNN)	Feature-size heuristic	Test reduction
Shu et al. [39] (Perf-AL)	Generative adversarial network(GAN)	Feature-size heuristic	Test reduction
Oh et al. [32]	Recursive searching	Random	Finding optimal configurations
Nair et al. [31] (FLASH)	Classification and Regression Trees (CART)	Feature-size heuristic	Finding optimal configurations
Valov et al. [44]	Support Vector Machine(SVM)	Feature-size heuristic	Test reduction
Bao et al. [4] (AutoConfig)	Random Forest	Weighted Latin hypercube sampling	Finding optimal configurations
Queiroz et al. [35]	Naive Bayes/Random Forest/C4.5	Arbitrarily chosen	Finding optimal configurations/Detecting anomalies
Lillack et al. [26]	Linear Regression	Feature-coverage heuristic /Feature-size heuristic	Finding optimal configurations
Chen et al. [7]	Linear Regression	Arbitrarily chosen	Test reduction
Siegmund et al. [41]	Feature's Influence Delta	Knowledge-wise heuristic	Test reduction/Increasing interpretability
Siegmund et al. [40]	Feature's Influence Delta	Feature-coverage heuristic /Feature-size heuristic	Test reduction/Increasing interpretability
Henard [16]	-	Solver based	-
Xiang et al. [45] (NSbS)	-	Solver based	Test reduction
Kaltenecker et al. [20]	SPL machine learning approach	Diversified distance based /SPL sampling approach	Test reduction
Luo et al. [27]	-	Genetic sampling	Detecting anomalies
Lemieux et al. [24]	-	Genetic sampling	-
Guo et al. [12]	-	Genetic sampling	Finding optimal configurations
Martinez et al. [29]	Data mining interpolation technique	Genetic sampling	Finding optimal configurations
Jamshidi et al. [18]	Step-wise multiple linear Regression	Random	Performance model reuse
Krishna et al. [23] (BEETLE)	Regression tree	Random	Performance model reuse

Detecting anomalies. Similar to searching for optimal configurations, another purpose of configuration performance modeling is to detect performance anomalies. Queiroz et al. [35] present a technique for finding anomalies and the optimal configuration using multiple modeling methods, such as Random Forest and C4.5.

B. Sampling approaches of configuration performance modeling

In this sub-section, we present the sampling approach in the state-of-art configuration performance modeling approaches. A summary of the approaches is presented in Table I.

Random sampling. Random sampling is the most prevalent sampling approach in the configuration performance field. Prior research widely uses random sampling in the implementation of their approaches. We can distinguish random sampling used in prior research as simple random sampling [15], [18], [23], [32] and feature-size heuristic sampling [11], [13], [14], [31], [39]. The shortcomings of random sampling are apparent. Random sampling cannot truly determine the selection of the samples by the developers, and the randomness of the selection may cause the instability of prediction.

Distance-based sampling. Distance-based sampling is a large set of sampling approaches that utilize different measurements to measure the distance between the configurations. All the distance-based sampling approaches leverage different selection patterns for covering all the configurations. The Manhattan distance [22] is particularly widely used in prior research [4], [20], [26], [40].

Solver-based sampling. In the context of solver-based sampling, the solver algorithm searches for the optimal combination of the configuration mathematically based on the deployed solvers. We can find the application of solver-based sampling in several previous works [16], [41], [45].

Genetic sampling. Genetic sampling is a search-based method that simulates the process of natural selection to find an optimal solution to a problem, which is widely used in performance testing [24], [27]. Such algorithm is also used for sampling in configuration performance modeling [12], [29].

C. The relationship between sampling approaches and purposes

In this sub-section, we discuss the relationship between the purpose and the sampling approaches for configuration performance models. In particular, the studies from prior research show that the sampling approaches are often driven by the purposes of the configuration performance modeling. For example, to find optimal samples or anomalies, the researchers can use a search-based method to search for the configuration with the best or worst performance [4], [12], [26], [29], [31], [32]. To address the need for model interpretability, developer require to comprehend all the features' connection from source code in the software for the initial building. Therefore, prior research may select samples by feature-coverage heuristics [40]. Similar to this paper, reducing performance tests is one of the most common purposes in prior studies. Almost all the sampling approaches are in use from prior research, depending on their particular purposes. For example, in order to cover all the features of the configurations in tests, distance-based or solver-based sampling approaches are leveraged [20], [45].

Our motivation. The sampling approaches from prior research do not focus on building a better performance model. These approaches rather aim to search for samples to increase the test coverage of the configurations. As a result, random sampling has been found to have a similar performance compared to other sampling approaches [14], [20]. Such findings from prior research indicate that the range of the sample

configurations may not truly reflect the performance behaviour of the model.

To address this limitation, we propose sampling approaches for better configuration performance models by measuring the uncertainties of each sample and leveraging historical performance measurement. Instead of trying to achieve a higher coverage on the combinations of the configurations, our approach directly aims to improve the accuracy of the configuration performance models that are built from these selected samples.

III. CoMSA: Our Modeling-Driven Sampling Approach

In this section, we present our approach named CoMSA (Configuration performance Modeling Sampling Approach) in three parts. We first describe the problem that we attempt to solve. Afterwards, we present our approach in two scenarios, the cold-start scenario, and the warm-start scenario. Our code is available in a supplementary website[1] for more details.

A. Problem statement

For a configurable software system, let $\mathcal{X}$ be the set of configurations formed by combining various configuration options. For each configuration $\mathrm{x} \in \mathcal{X}$, it usually leads to a certain measurement value that indicates performance (e.g., response time), i.e., $\mathrm{y} \in \mathcal{Y}$. In this paper, our main task is to propose a selector S that can effectively select m configurations to measure/test (i.e., $\mathcal{X}_M \subseteq \mathcal{X}$). Based on the measured performance of $\mathcal{X}_M$, combined with prior knowledge of the system performance, we train a performance model to accurately predict the performance of the unmeasured/untested configurations.

Formally, the optimization problem of the configuration selection can be expressed as follows:

$$\underset{\{\mathcal{X}_M,\mathcal{Y}_M\}\subseteq\{\mathcal{X},\mathcal{Y}\}}{\arg\min} \quad Error_{(x,y)\in\{\mathcal{X}_U,\mathcal{Y}_U\}}\left[f_\theta\right] \tag{1}$$

where, $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ is the model with parameters, θ, trained on the measured configurations, $\{\mathcal{X}_M, \mathcal{Y}_M\}$, which is selected by a selector S, and $Error$ is the error between the actual performance and the predicted performance on the unmeasured configurations, $\mathcal{X}_U = \mathcal{X} \setminus \mathcal{X}_M$ and $|\mathcal{X}_M| \ll |\mathcal{X}_U|$. The goal is to select m most appropriate configurations as the training set to ensure the trained performance model has the smallest error on the unmeasured configurations. In the following subsections, we introduce the approaches on how to select m configurations to build the training set.

B. Cold-start scenario

A software project (especially the new one) may not have a historical repository of its performance testing results. For a project that would like to start testing the performance of different configurations, one first needs to select a sample of configurations for performance testing. We call such a scenario as *Cold-start* scenario.

[1]https://github.com/Yuanjie-Xia/CoMSA

In the cold-start scenario, different sampling approaches (cf. Section II-B) have been proposed to select a representative set of configurations for training a performance model. However, sampling approaches from prior research barely consider the uncertainty of the configurations with regard to the performance model. Intuitively, the sample selection process and the performance model should be influenced by each other. On one hand, different selections of configurations can lead to different performance models. On the other hand, to have more accurate prediction results, different models may favour different configurations for training, and thus the model should also guide the configuration selection process, e.g., selecting the most uncertain configurations (i.e., of which the performance is difficult to predict) to measure.

Therefore, we propose to model the uncertainty of the configurations, w.r.t. the performance model, f_θ, by employing an ensemble approach where a collection of δ models (i.e., $f_1, \ldots, f_\delta$, $\delta \geq 2$) are trained independently. The δ models constitute the selector S, which has the same architecture with f_θ, but different parameters. Below we discuss the details of how we build the selector S (i.e., a collection of δ models with different parameters) and select the most uncertain configurations to measure.

The overview of the configuration selection process is shown in Figure 1 Part 1, where we follow an active learning paradigm to iteratively select the most uncertain configuration.

Step 1.1: Selector initialization. To initialize our selector S (i.e., train an ensemble of δ models, $f_1, \ldots, f_\delta$), we first randomly select δ distinct sets of configurations as the training data. By using different training sets, we can have δ models with the same architecture, but different parameters. Note that the initial training set is relatively small for a more convincing evaluation since our approach tends to reduce the influence from random selection(i.e., a total of six configurations for LRZIP, LLVM, three for x264, and 15 for SQLite, respectively.).

Step 1.2: Uncertainty measurement. After having the selector S, we start to select the most uncertain configuration for further measurement. Specifically, we apply the δ trained models on the unmeasured configurations and predict the performance separately. As a result, we have δ predictions for each unmeasured configuration. We use the standard deviation as a measure of the uncertainty for each configuration. The configuration of which the predictions have the highest standard deviation is considered as the most uncertain configuration and then is selected for measurement.

Step 1.3: Selector update. We augment each of the δ training sets with the newly measured configuration, and update the selector S by re-training each of the δ models with the δ augmented training sets, respectively.

Step 1.4: Performance model training. We repeat Step 1.2 and Step 1.3 until we reach the pre-defined measurement budget (i.e., a total of m measurements). Finally, we use all the m measurements as the data set for training a performance model f_θ.

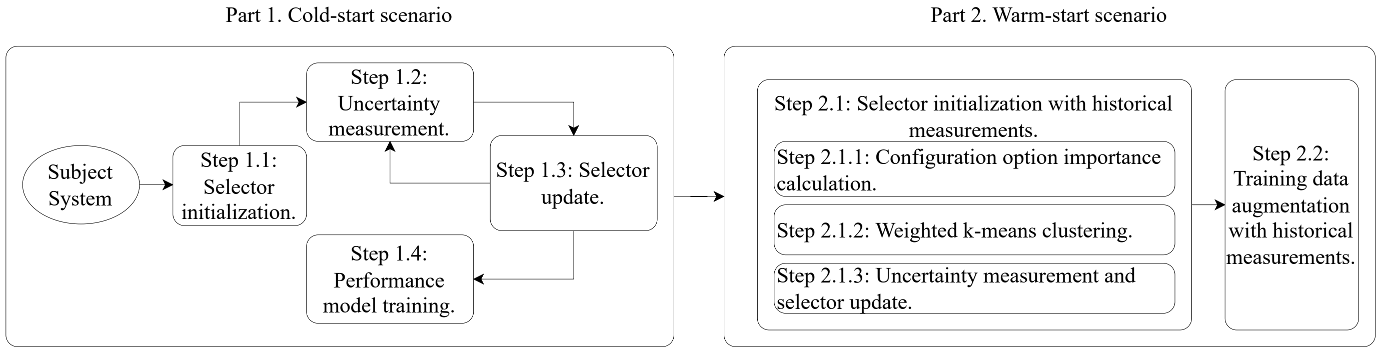


Fig. 1: Overview of approach.

C. Warm-start scenario

In the previous subsection (i.e., cold-start scenario), given a limited budget for performance testing (i.e., a maximum number of m configurations to measure) of a certain version of a software system, we propose to select the most uncertain configurations w.r.t. the performance model, aiming to reduce the prediction errors on the unmeasured configurations.

However, software systems usually evolve rapidly with frequent code commits, the performance testing would be costly if we still measure a group of m configurations for each commit. On the other hand, historical performance testing results are typically available during the software quality control process. Although a commit may change the source code and impact the performance (i.e., the same configuration has different performances between two consecutive commits), with frequent testing in each commit, such impact can be alleviated by only measuring a relatively small set of configurations of the new commit. Therefore, it is intuitive to consider reusing the historical measurements from previous commits to reduce the number of configurations that need to be measured for new commits, while also maintaining an acceptable level of prediction accuracy [18], [23], [28], [30]. We call such a scenario a *Warm-start* scenario.

In this subsection, we propose an information augmentation-based approach to introduce additional information 1) not only into the configuration sampling process, 2) but also into the performance model training, aiming to reduce the measurement cost of building an accurate performance model. The overview of this part is shown in Figure 1 in Part 2.

Step 2.1: Selector initialization with historical measurements. In the cold-start scenario (cf. Section III-B), we initialize the selector S in a random manner. However, the randomness of the selector initialization imposes a non-negligible effect on the following iterative new configuration selection process, making the trained performance model unstable, especially when the sample size is small.

Therefore, in this part, to avoid the (negative) effects of random initialization and better guide the selection of configurations of a new commit, we utilize historical measurements to initialize the selector in a warm start manner. Specifically, in the previous section, we have measured a total of m configurations. Instead of simply separating the m measurements into δ distinct sets as the training data for selector initialization, we use weighted k-means clustering to divide the historical measurements into δ distinct training sets. Below, we detail the steps for constructing the initialization sets and updating the selector:

Step 2.1.1: Configuration option importance calculation. Considering the different impacts of configuration options on the software's performance, we calculate the importance of each configuration option based on the feature importance of the trained model, which is used as the weight for clustering. In our experiments, we use the performance model trained in the previous section to extract the importance of each option.

Step 2.1.2: Weighted k-means clustering. We then perform clustering on the collected historical measurements based on the weighted distance of the configuration options. Following previous work [4], [20], we use the Manhattan distance to measure the distance between configurations. Finally, the historical measurements are clustered into δ groups for initializing the selector, of which δ different models are trained.

Step 2.1.3: Uncertainty measurement and selector update. Once we have the initialized selector, we follow Steps 1.2 and 1.3 to iteratively select the most uncertain configurations for the new commit. Note that when the newly selected configuration appears in the historical measurements, we replace the historical measurement with the new measurement.

Step 2.2: Training data augmentation with historical measurements. Besides initializing the selector with the historical measurements, we also adopt the transfer-based data augmentation strategy to enlarge the newly measured configurations. Specifically, during the performance model training step, we combine all the historical measurements and the newly measured configurations as the data set for training a performance model f_θ.

By utilizing the historical measurements for both the selector initialization and performance model training, we aim to reduce the number of configurations needed for measuring the new commit, while still maintaining an acceptable level of accuracy of the trained performance model.

TABLE II: An overview of our subject systems.

Subject	Domain	$\|\mathcal{B}\|$	$\|\mathcal{N}\|$	$\|\mathcal{C}\|$	Performance value	$\|\mathcal{H}\|$
LRZIP	File Archive Utility	5	4	1000	Compression time	61
LLVM	Compiler	9	0	512	Running time	55
x264	Video Encoder	16	2	1535	Encoding time	48
SQLite	Database System	20	5	3456	Response time	52

$|\mathcal{B}|$: the number of the binary configuration options. $|\mathcal{N}|$: the number of numeric configuration options. $|\mathcal{C}|$: Total number of the configurations. $|\mathcal{H}|$: number of the used commits.

IV. Evaluation Setup

In this section, we present the setup of the evaluation of our approach.

A. Subject systems

To evaluate our proposed approach, we consider four highly configurable software systems: LRZIP, LLVM, x264 and SQLite. We choose the four subject projects for several reasons: 1) they are from different domains, including file archive utility, compiler, video encoder, and database, 2) the four selected systems have been widely used in previous research [14], [20], and 3) they have been rapidly evolving over a long period of time and provided a complete commit history tracking changes and updates. Table II further details each considered subject system, in which the values of the binary options in the same configuration are inversely dependent on each other.

B. Experimental environment

For LRZIP, LLVM and x264, we measure their performance on a virtual machine with 4 vCPU and 8 GB memory, and for SQLite, as it requires more memory, we use a virtual machine with 2 vCPU and 16 GB memory. All experiments are conducted on Amazon Web Services (AWS). To reduce the influence of the noise during the measurement of performance, we measure the performance of each configuration 10 times and then use the average as the final performance of the configuration.

C. Evaluation metrics of performance models

Our sampling approach CoMSA is a modeling-driven approach, i.e., the purpose of CoMSA is to build better performance models. Therefore, to evaluate the corresponding performance model, we utilize two metrics, R-Squared (R^2), and Root Mean Square Error (RMSE) [17]. Both of them have been commonly used for evaluating the performance of a regression model [10], [37].

R^2 is a statistical measure used to assess how well the unseen samples (in our context, the unmeasured configurations) are likely to be predicted by the model and provides an indication of the goodness of fit. In general, the higher the R^2, the better the model fits the data. RMSE is another commonly used metric for evaluating performance models [10]. RMSE computes the deviation of the prediction from the actual value, which quantifies the difference between the prediction and the actual value directly. A lower RMSE indicates a smaller error and higher prediction accuracy.

V. Preliminary Study

Before the evaluation of our approach, in this section, we aim to conduct a preliminary study by answering the following two preliminary questions (PQs). The answer to the two PQs further guides the evaluation of our approach.

PQ1: Are selected uncertain configurations specific to the performance models?

Motivation. If the uncertain configurations that are selected by our approach are not specific to the performance models, the choice of performance models would not be important in our evaluation. Otherwise, we need to carefully select the performance models used in our evaluation.

Approach. We apply our sampling approach CoMSA in a cold start manner on the latest version of all subject systems (at the time of the study) with four different performance models, i.e., Random Forest (RF), XGBoost, DeepPerf, and Perf-AL. Random Forest and XGBoost are traditional machine-learning methods that are widely used for many regression tasks. DeepPerf and Perf-AL are recent advanced methods, which achieve higher prediction accuracy than others[2] [11], [13], [44]. Following prior research [14], [45], we conduct the experiment with different configuration sizes (i.e., $m \in \{n, 2n, ...\}$, where n is the number of the binary configuration options.). Meanwhile, we repeat the experiments 20 times with different random seeds to avoid the influence of the randomness of selector initialization.

We count the overlapping configurations that are selected by CoMSA among the performance models. The lower the number of overlapping configurations that are selected by CoMSA, the more specific the selected configurations are to the performance models.

Result. **Different models may select different uncertain configurations.** The last column in Table III shows the average number of overlaps of configurations over the 20 runs. We find that the overlap is relatively small across different performance models. For example, when selecting n configurations, the average number tends to be zero for almost all the performance models. The small number of overlaps demonstrates the high specificness of the selected configurations for each model. The results indicate that when building a performance prediction model, we should consider selecting different configurations for different modeling methods. More pragmatically, the results of this PQ lead us to PQ2, which concerns which performance model provides the best modeling results.

PQ2: How do different models perform with our sampling approach?

Motivation. The results of PQ1 show that the uncertain configurations that are selected by our approach are specific to the performance models. Therefore, we need to select the best performance models to use in our evaluation.

Approach. We evaluate the performance of different models with our sampling approach. We use R^2 and RMSE, which are presented in Section IV-C as the evaluation metrics.

[2]For XGBoost, Random Forest (RF), and Perf-AL, we use the default parameters and perform parameter tuning on DeepPerf.

TABLE III: Details of the mean of R^2, RMSE, and overlapping configurations over the 20 repeated experiments using different models.

Subject	Sample size (%)	RF		XGBoost		DeepPerf		Perf-AL		Overlap
		RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	
LRZIP	n (0.9%)	123.71	**0.32**	**117.46**	0.04	120.21	0.17	159.68	0.28	0
	2n (1.8%)	47.62	0.90	**41.37**	**0.92**	77.38	0.84	128.22	0.26	0.4
	3n (2.7%)	24.76	0.92	**18.46**	**0.98**	70.35	0.89	113.13	0.36	0.4
	4n (3.6%)	19.48	0.93	**15.85**	**0.99**	53.11	0.89	113.27	0.25	1.8
	5n (4.5%)	18.84	0.93	**10.93**	**0.99**	35.97	0.96	95.02	0.58	2
	6n (5.4%)	18.69	0.93	**10.77**	**1.00**	29.06	0.99	110.08	0.77	2.4
LLVM	2n (3.9%)	1.52	-0.02	**1.41**	**0.08**	1.60	-0.15	1.77	-0.21	0.6
	4n (7.8%)	1.30	0.13	**1.26**	**0.25**	1.45	0.18	1.68	-0.03	2.6
	6n (11.7%)	1.28	0.27	**1.19**	**0.32**	1.42	0.19	1.61	0.04	2.8
	8n (15.6%)	1.22	0.32	**1.13**	**0.42**	1.28	0.36	1.59	0.07	3.8
	10n (19.5%)	1.16	0.34	**1.07**	**0.48**	1.20	0.44	1.58	0.16	6.2
	12n (23.4%)	1.13	0.37	**1.03**	**0.52**	1.20	0.45	1.55	0.16	7.8
	14n (27.3%)	1.07	0.38	**0.99**	**0.54**	1.14	0.38	1.54	0.24	10
	16n (31.3%)	1.02	0.37	**0.96**	**0.56**	1.10	0.49	1.47	0.23	16.8
	18n (35.2%)	1.01	0.39	**0.94**	**0.58**	1.09	0.42	1.44	0.22	18.6
	20n (39.1%)	0.99	0.40	**0.91**	**0.60**	1.04	0.52	1.51	0.25	25.8
x264	0.5n (0.3%)	17.43	0.09	**11.36**	**0.70**	15.65	0.41	25.42	-0.30	0
	n (0.7%)	10.16	0.66	**6.82**	**0.90**	9.73	0.79	21.93	-0.05	0
	1.5n (1.0%)	6.70	0.81	**4.20**	**0.94**	5.27	0.85	19.87	0.35	0.2
	2n (1.3%)	5.59	0.85	**3.56**	**0.95**	3.76	0.92	18.35	0.34	0.2
	2.5n (1.6%)	5.18	0.86	2.96	**0.96**	**2.47**	0.95	18.03	0.27	0.6
	3n (2.0%)	5.06	0.86	2.59	0.96	**2.34**	**0.99**	16.82	0.38	0.8
	3.5n (2.3%)	4.92	0.87	2.46	0.96	**2.10**	**0.99**	15.88	0.54	1.2
	4n (2.6%)	4.72	0.87	2.34	0.97	**1.55**	**0.99**	15.86	0.52	1.2
SQLite	n (0.7%)	**9.73**	-0.19	9.92	**0.84**	10.13	0.11	25.50	-0.37	0
	2n (1.4%)	2.99	0.71	4.98	**0.99**	**4.58**	0.54	24.01	-0.41	0.2
	3n (2.2%)	2.26	0.96	2.79	**1.00**	**2.19**	0.74	23.09	-0.23	0.4
	4n (2.9%)	2.21	0.97	**1.49**	**1.00**	1.81	0.85	21.22	-0.11	0.8
	5n (3.6%)	1.94	0.97	**1.05**	**1.00**	1.46	0.99	18.87	-0.06	1.8

Result. **XGBoost has an overall best result among the four evaluated modeling approaches.** Table III shows the results of Random Forest, XGBoost, DeepPerf, and Perf-AL, with the best results highlighted in bold. By comparing XGBoost with the other three models, we find that XGBoost achieves better performance for most of the subject systems across different configuration sizes. For example, in terms of R^2, XGBoost always has the best performance for systems LRZIP, SQLite, and LLVM. Besides, as shown in Table IV, we also find that XGBoost requires less training time, which makes it more suitable to be adopted in practice. These results also confirm the previous findings that simpler baselines run faster and may outperform complex techniques [9], [21].

TABLE IV: Training time of different performance models.

Models	Training time (seconds)
Random Forest	2.49
XGBoost	2.18
DeepPerf (boost with GPU)	111.57
Perf-AL (boost with GPU)	33.20

VI. Evaluation

In this section, we present a detailed evaluation of our approach CoMSA. In particular, we evaluate CoMSA in the cold-start and the warm-start scenarios, respectively.

A. Evaluation of CoMSA in the cold-start scenario

In this subsection, we evaluate CoMSA in comparison with other sampling approaches in the cold-start scenario.

Motivation. As presented in Section II, many configuration sampling approaches have been proposed to select representative configurations for building a performance prediction model in the cold-start scenario. We then propose CoMSA to select the most uncertain configurations for measurement, aiming to improve the model performance on the unmeasured configurations. Therefore, in this section, we would like to explore whether our modeling-driven approach CoMSA can have a better performance on the unmeasured configurations than the existing sampling approaches.

Approach. Based on the findings from our preliminary study (cf. Section V), we opt to use XGBoost in our evaluation as the modeling method. We select a total of nine sampling approaches as baselines, including distance-based sampling, solver-based sampling, etc. Similar to the evaluation approach in Section PQ1, we conduct experiments with different configuration sizes and repeat the experiments 20 times. We compare the performance of CoMSA with baselines using R^2 and RMSE.

To provide a more comprehensive evaluation, we also utilize the ScottKnott Effect Size Difference (ESD) test [42] to cluster and rank the performance of all the sampling approaches. The ESD test incorporates multiple statistical parameters to provide a more objective result. In particular, The ESD test cluster the approaches using the mean and sum of squares. Then, it utilizes the likelihood ratio test, Chi-square distribution, and Cohen's effect size to split and merge the cluster. Through the ESD test, we can obtain the clusters of approaches that have a set of approaches with negligible differences and the ranking of the clusters.

To further understand our results, we also calculate the relative standard deviation (RSD) of each subject system performance measurements [3], which is widely used to estimate the variance of a dataset [36], [43]. A higher RSD means that the values of the measurements are more widely spread from the average, and thus more difficult to model [5]. We discuss our results by considering the RSD of each subject.

Results. **CoMSA builds better performance models compared to all baselines in general.** The results of the sampling approaches are shown in Table V. Based on the results, we find that the performance model built with CoMSA exhibits higher R^2 and lower RMSE compared to other baseline approaches in general. Specifically, CoMSA has the highest R^2 in 21 out of 30 experiments, which is much larger than the second best-performing approaches, FLASH and Gentic. The result demonstrates the high suitability of using a modeling-driven approach such as CoMSA to build better configuration performance models with smaller samples.

Other sampling approaches may sometimes have slightly better results than CoMSA. For example, the Genetic method has better performance when the number of samples is small in LRZIP and LLVM. The genetic sampling approach aggressively mutates samples, leading to diverse performance model results. Hence, some high-performing samples that are selected by the genetic search may have a significant influence on the mean R^2 value, particularly in cases with small sample sizes.

TABLE V: Details of the mean of R^2 and mean of RMSE among all repeated tests of using different selection approaches in different subjects.

Subject	Sample size	CoMSA		Random		Genetic [27]		DistBased [45]		divDistBased [45]		henard [16]		NsbS [45]		solverBased [6]		coverBased [40]		FLASH [31]	
		RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
LRZIP	n	117.46	0.04	138.64	0.29	146.99	**0.70**	89.26	-0.08	136.13	0.00	97.73	0.53	132.77	-0.00	95.78	0.07	**76.47**	-0.22	173.71	0.16
	2n	41.37	**0.92**	86.67	0.90	48.15	0.87	82.28	0.39	54.76	0.82	**37.19**	0.71	67.67	0.59	70.01	0.70	67.33	-0.23	96.53	0.63
	3n	**18.46**	**0.98**	71.18	0.92	33.68	0.94	102.47	0.38	37.94	0.82	37.33	0.78	44.14	0.51	59.75	0.78	80.14	-0.23	98.34	0.59
	4n	**15.85**	**0.99**	55.11	0.95	29.08	0.94	64.92	0.52	21.50	0.93	28.07	0.90	29.42	0.59	44.90	0.75	56.40	-0.24	129.30	0.25
	5n	**10.93**	**0.99**	46.83	0.96	27.06	0.94	63.58	0.57	13.81	0.94	18.88	0.92	15.77	0.62	31.56	0.76	75.32	-0.24	157.39	-0.10
	6n	**10.77**	**1.00**	25.47	0.99	26.22	0.94	64.22	0.63	12.01	0.95	24.51	0.97	15.42	0.69	35.41	0.76	75.16	-0.25	138.99	0.01
LLVM	2n	1.41	0.08	**1.35**	0.10	1.43	**0.11**	2.36	-0.05	2.33	-1.44	2.15	-5.09	2.74	-1.33	2.61	-0.95	2.22	-1.95	1.57	-1.06
	4n	1.26	0.25	**1.24**	0.15	1.41	**0.27**	2.57	0.08	2.55	-2.05	2.65	-4.05	2.68	-1.97	2.25	-2.23	2.38	-1.22	1.46	-1.54
	6n	1.19	0.32	**1.16**	0.29	1.42	**0.36**	2.15	0.13	2.18	-1.17	2.29	-2.81	2.44	-1.25	2.08	-1.34	2.30	-0.94	1.41	-1.30
	8n	1.13	**0.42**	**1.11**	0.36	1.39	0.41	1.85	0.14	1.80	-0.59	2.45	-2.54	2.41	-0.53	1.86	-1.89	2.18	-0.57	1.39	-0.99
	10n	1.07	**0.48**	**1.05**	0.43	1.35	0.45	1.49	0.10	1.29	-0.03	1.97	-2.56	2.44	0.22	1.72	-0.90	2.26	-0.35	1.41	-1.29
	12n	**1.03**	**0.52**	1.04	0.44	1.30	0.46	1.48	0.16	1.30	0.01	1.95	-2.97	2.49	0.22	1.45	-0.79	2.45	0.04	1.35	-2.02
	14n	**0.99**	**0.54**	1.02	0.45	1.28	0.48	1.45	0.19	1.32	0.06	1.61	-1.64	2.13	0.20	1.34	-0.32	2.44	0.18	1.31	-2.02
	16n	**0.96**	**0.56**	1.00	0.47	1.26	0.50	1.25	0.21	1.15	0.28	1.68	-1.37	2.07	0.38	1.32	-0.38	2.44	0.24	1.30	-2.02
	18n	**0.94**	**0.58**	0.98	0.48	1.23	0.50	1.27	0.22	1.16	0.24	1.57	-1.20	2.06	0.35	1.14	-0.20	2.44	0.42	1.29	-2.02
	20n	**0.91**	**0.60**	0.98	0.49	1.20	0.51	1.27	0.22	1.16	0.24	1.57	-1.14	2.07	0.35	1.14	-0.20	2.44	0.42	1.29	-2.02
x264	0.5n	**11.36**	**0.70**	17.63	-0.79	12.81	0.55	20.74	0.69	17.96	-0.20	22.60	-0.77	27.34	0.02	26.73	-0.34	30.30	-0.71	14.02	-1.17
	n	6.82	**0.90**	9.57	0.53	9.24	0.72	16.70	0.70	9.49	0.13	**6.74**	-0.39	22.89	0.70	11.37	0.89	30.71	0.52	13.36	-1.25
	1.5n	**4.20**	**0.94**	6.36	0.84	7.80	0.78	16.21	0.70	7.25	0.11	6.18	0.14	16.59	0.85	11.28	0.91	30.23	0.52	11.13	-1.18
	2n	**3.56**	**0.95**	5.04	0.91	5.91	0.85	14.02	0.70	5.88	0.28	5.99	0.41	12.97	0.91	8.09	0.91	27.96	0.78	11.07	-0.87
	2.5n	**2.96**	0.96	4.06	0.93	5.65	0.86	11.01	0.70	4.46	0.48	6.06	0.77	7.68	0.94	8.03	0.91	3.61	0.78	11.09	**0.97**
	3n	2.59	0.96	3.41	0.96	5.37	0.87	8.55	0.70	3.57	0.69	5.94	0.89	5.66	0.96	7.52	0.91	**2.08**	0.80	11.00	**0.99**
	3.5n	2.46	0.96	3.08	0.97	4.74	0.88	6.03	0.70	3.13	0.82	5.97	0.95	3.76	0.97	7.51	0.91	**2.14**	0.80	11.02	**0.99**
	4n	2.34	0.97	2.70	0.98	4.32	0.89	4.85	0.70	2.82	0.86	6.00	0.96	3.68	0.98	7.96	0.91	**1.56**	0.79	11.04	**0.99**
SQLite	n	**5.58**	0.84	9.92	-0.53	14.00	0.41	10.86	0.82	9.99	-1.01	10.56	**0.87**	6.90	0.73	10.93	0.74	10.84	0.36	7.75	0.71
	2n	**2.02**	**0.99**	4.98	0.79	13.52	0.43	11.04	0.90	8.56	0.56	2.99	0.96	2.96	0.79	10.96	0.98	9.73	0.55	5.53	0.76
	3n	1.23	**1.00**	2.79	0.94	12.20	0.46	10.65	0.96	7.21	0.50	**1.14**	**1.00**	1.15	0.83	8.61	0.99	9.76	0.63	3.27	0.76
	4n	0.95	**1.00**	1.49	0.97	11.89	0.47	8.69	0.97	4.53	0.56	1.10	**1.00**	**0.87**	0.91	7.11	0.99	9.68	0.67	2.84	0.77
	5n	0.91	**1.00**	1.05	0.99	11.84	0.47	7.24	0.98	3.10	0.62	1.04	**1.00**	**0.80**	0.96	5.57	1.00	9.76	0.76	2.09	0.76

The negative R^2 can cause by overfitting or extremely poor training.

However, with the increase in sample sizes, the negative effect of genetic sampling would start to appear. Therefore, due to the selector random initialization, we do see that CoMSA may not build the best performance models when randomly starting the first iteration of small samples. This finding helps us recognize the importance of selector initialization and motivate us to design a more reliable selector initialization (cf. Section III-C step 2.1) by utilizing historical measurements.

Although there are a few cases where other sampling approaches have better performance than CoMSA, the differences are small. For example, the FLASH sampling approach [31] achieves the best performance in terms of R^2 for project x264 when the sample size is larger than 2.5n, but the performance gap is negligible compared to that of CoMSA (e.g., 0.96 vs. 0.97 when size is 2.5n). However, when the sample size is smaller than 2.5n for the same project x264, the FLASH even has a negative R^2. The diverse optimal values of the baseline sampling approaches indicate that these sampling approaches have high instability in building high-accuracy performance models and further demonstrate the dominance of CoMSA across different projects and configuration sizes.

CoMSA remains at the Top 3 list in ScottKnott Effect Size Difference (ESD) test[3]. Table VI shows the Top-3 ranking results of the ESD test. Although some sampling approaches may perform better in specific conditions, we can observe that CoMSA consistently remains in the Top 3 list.

[3]Due to the limited space, we only show the results in R^2. The results in RMSE is shared in our replication package.

TABLE VI: Top 3 clusters in the ScottKnott Effect Size Difference (ESD) test.

Subject	Sample size	Rank of sampling approaches		
		Top 1	Top 2	Top 3
LRZIP	n	Genetic	CoMSA	divDistBased
	2n	CoMSA	Genetic	FLASH
	3n	CoMSA	Genetic	FLASH
	4n	CoMSA	Genetic	divDistBased
	5n	CoMSA	Genetic	divDistBased
	6n	CoMSA	Nsbs	divDistBased
LLVM	2n	CoMSA	Genetic	Random
	4n	CoMSA	Genetic	Random
	6n	CoMSA	Genetic	Random
	8n	CoMSA	Genetic	Random
	10n	CoMSA	Genetic	Random
	12n	CoMSA	Random	Genetic
	14n	CoMSA	Genetic	Random
	16n	CoMSA	Genetic	Random
	18n	CoMSA	Genetic	solverBased
	20n	CoMSA	Genetic	solverBased
x264	0.5n	FLASH	divDistBased	CoMSA
	n	solverBased	Genetic	CoMSA
	1.5n	CoMSA	Genetic	divDistBased
	2n	CoMSA	Genetic	divDistBased
	2.5n	CoMSA	divDistBased	NsbS
	3n	coverBased	CoMSA	divDistBased
	3.5n	coverBased	CoMSA	divDistBased
	4n	coverBased	CoMSA	divDistBased
SQLite	n	CoMSA	NsbS	Genetic
	2n	CoMSA	divDistBased	NsbS
	3n	CoMSA	NsbS	henard
	4n	CoMSA	NsbS	henard
	5n	NsbS	CoMSA	henard

In particular, CoMSA always ranks in the Top 1 for LRZIP and LLVM. Unlike other approaches, the CoMSA method demonstrates higher stability across different subjects and sampling sizes.

CoMSA builds better performance models than the baseline, especially with more widely spread performance measurements. We calculate the RSD of our four subject systems. In particular, the RSD values of LRZIP, LLVM, x264, and SQLite are 185, 36, 78, and 19, respectively. The RSD values show that the performance of LRZIP is mostly diversified, potentially the most difficult to model, while SQLite is the easiest. In fact, CoMSA performs very well when building performance models for LRZIP, which has the highest RSD. The high variance of performance among different configurations causes higher uncertainty measurements for the selector's update, which helps CoMSA achieve a higher accuracy due to the awareness of uncertainty by CoMSA. On the other hand, in SQLite, we can observe that the low RSD may cause more baseline sampling approaches also to perform well. Even in such cases, CoMSA still outperforms the baseline sampling approaches by achieving nearly optimal performance modeling results.

> CoMSA presents an outstanding performance in the cold-start scenario than the baseline approaches. The result demonstrates the high suitability of using modeling-driven approaches such as CoMSA to build better configuration performance models with smaller sample sizes.

B. Evaluation of CoMSA in the warm-start scenario

In this subsection, we evaluate CoMA in the warm-start scenario. In other words, we would like to see if CoMCA can leverage historical measurements from previous commits to reduce the number of samples required for new commits while maintaining the accuracy of performance models.

Motivation. In the last subsection, we evaluate CoMSA in the cold-start scenario that mainly serves applications that do not store historical performance testing results. Once it is possible to obtain historical performance testing results, i.e., warm-start scenario, we would like to explore whether we can possibly reduce the performance testing efforts for a rapidly evolving software environment with such historical information.

Approach. To evaluate the effect of utilizing historical measurements in different ranges, we divide our evaluation into two steps: 1) using historical measurements from only one past commit, and 2) using all the historical measurements.

Using only one most recent commit. Intuitively, the performance of software from the one most recent commit may be the most similar to the current commit. Therefore, we first only leverage the information from the one most recent commit in our warm-start scenario.

Before starting our experiment in the warm-start scenario, we use the performance models that are built in the cold-start scenario (see Table V) to determine the acceptable level of accuracy for the performance model of a new commit. To determine such accuracy, we search for the inflection points of performance in each subject. Mathematically, the inflection point can be defined as the point where the second derivative of a function changes its sign, indicating a decrease in the rate of increase of R^2. Based on the evaluation results in Section VI-A, we identified the inflection point of LRZIP in $2n$, LLVM in $4n$, x264 in $1.5n$, and SQLite in $2n$.

TABLE VII: The mean of required configurations over all commits in the cold-start and warm-start scenarios.

Subject	Results of cold-start scenario	Mean (95th percentile) of results of warm-start scenario		
		Original	w/o selector initialization	w/o training data augmentation
LRZIP	18	3.78 (6)	16.65 (25)	5.93 (18)
LLVM	40	0.32 (4)	33.15 (35)	1.23 (27)
x264	15	0.11 (4)	6.63 (11)	0.49 (8)
SQLite	50	0.06 (17)	29.09 (46)	16.70 (36)

For each commit, we leverage our approach to iteratively select samples of configurations, until the performance models built from these samples have a higher R^2 than the values identified in the inflection points of each subject. For example, for LRZIP, for each commit, we keep adding samples of configurations until the R^2 of the model surpasses the R^2 value at the inflection point sample size. Our goal is to evaluate, to achieve a performance model with similar accuracy as in the cold-start scenario, how many samples we can save if we just consider the performance test results from one most recent commit. Therefore, we compare the number of samples needed in the warm-start scenario (based on only one recent commit), to the number of samples needed in the cold-start scenario.

Furthermore, we have designed two ablation experiments to assess the impact of utilizing historical measurements on 1) selector initialization and 2) training data augmentation. The first ablation experiment disables the selector's initialization with historical measurements (i.e., Step 2.1 in Section III-C). In addition, another ablation experiment disables the training data augmentation with historical measurements (i.e., Step 2.2 in Section III-C).

Using all historical commits. One may also consider using all historical performance testing results from all historical commits in the warm-start scenario. In particular, starting from the second commit of each subject (since the first commit does not have historical information), we use all the performance testing results from their prior commits with the approach described in Section III-C. In order to determine the number of samples needed for each commit, we opt for an overestimation by choosing a very high number of samples based on the results from our last step (using only one most recent commit). We choose the 95th percentile (almost the highest) of the number of samples of all commits of each subject as the number of samples needed for each commit. We use such an overestimation to avoid bias benefiting the warm-start scenario results. Finally, we use R^2 to compare the cold-start scenario and warm-start scenario that leverages all historical commits.

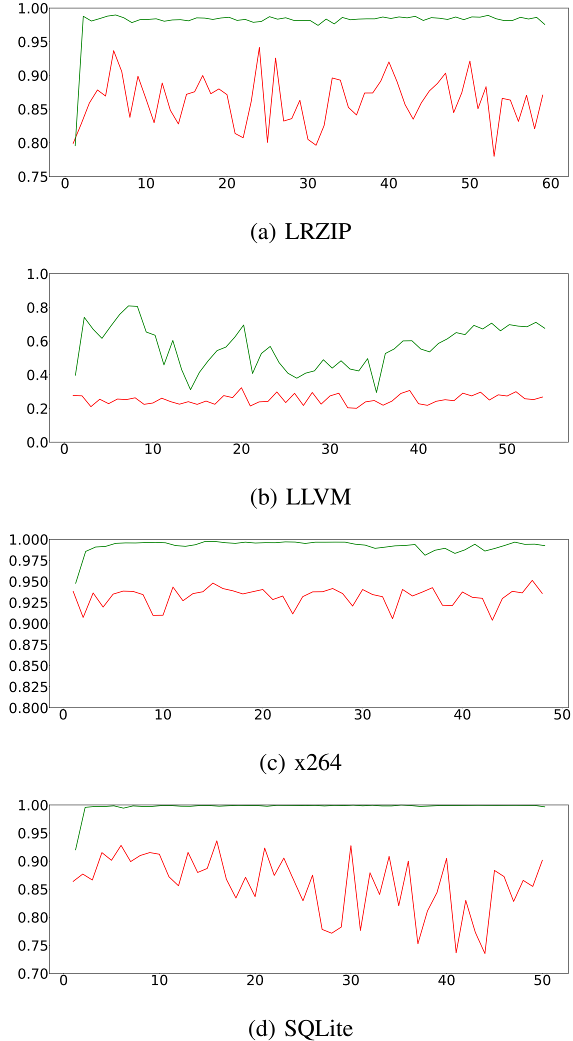
Results of using one most recent commit. **CoMSA can further reduce configuration performance testing efforts with historical measurements.** Table VII shows that in the warm-start scenario, considering only one most recent commits, one

can already expect a high reduction in the needed samples for configuration performance testing. In particular, in many cases, the mean number of needed tests is nearly zero, meaning that just adopting the existing performance data can already build a good performance model. When considering a more extreme case by looking at the 95th percentile, we can still have up to 90% reduction compared with the cold-start scenario. In addition, in the results of cold-start scenarios (see Table V, we can see that due to the random samples picked in the first iteration in cold-start scenarios, the models built with the initial samples are often with low accuracy; while in the warm-start scenario, we can avoid having a random sample and can start the iterations with already a relatively accurate model.

Both selector initialization and training data augmentation with historical measurements play important roles in reducing the samples for configuration performance models. Table VII presents the results of the ablation experiments. We find that either the selector initialization step or the training data augmentation step would result in a negative impact on our results. For example, without the selector initialization step, CoMSA would still retest some samples that have been tested in the past commit even though the configuration performance of these samples has negligible changes. For example, in the case of LRZIP without selector initialization, the numbers of configurations needed to be tested in the new commits are even closer to the cold-start scenario, illustrating that the sampling approach waste too much time on rebuilding the performance model. Similarly, without training data augmentation, some subjects, such as LLVM, SQLite, also experience a significant increase in the number of configurations needed to be tested in the new commits. Compared with the impact of selector initialization, the effect of training data augmentation is relatively smaller, highlighting the bigger importance of selector initialization. The effectiveness of the selector initialization in the warm-start scenario complements the results where CoMSA may not have optimal results with small initial random samples in the cold-start scenario.

> In the warm-start scenario, considering only one most recent commit, CoMSA can already reduce the number of samples required for new commits. The ablation experiments illustrate the importance of both steps from CoMSA in the warm-start scenario.

Results of using all historical commits. **Using more historical commits further improves the accuracy of performance models with fewer needed samples, compared with the cold-start-scenario.** Figure 2 provides a visualization of the improvement in performance model accuracy[4] for each commit, comparing the accuracy of the performance model with and without historical measurements. The green line represents the accuracy of the performance model that utilizes



(a) LRZIP

(b) LLVM

(c) x264

(d) SQLite

Fig. 2: The comparison between using historical measurements cumulatively (green line) and without using historical measurements (red line). The horizontal axis is the commit and the vertical axis represents the R^2.

all historical measurements for every commit, while the red line represents the accuracy of the performance model built in the cold-start scenario. Our results show that in the cold-start scenario, the accuracy of performance models can vary significantly even with the same number of samples across different commits. On the contrary, in the warm-start scenario with all historical measurements can maintain a consistently higher performance accuracy across commits.

Compared with the stable accuracy of the performance models in LRZIP, x264 and SQLite, the results of LLVM are rather unstable. We consider the reason for the instability is the relatively lower accuracy in the performance models built for LLVM. We find in the cold-start scenario that the performance models built for LLVM have R^2 values around 0.6 (see Table V); while other subjects' R^2 are much higher. Even with such instability of LLVM, the models build in the

[4]Due to the limited space, we only show the results in R^2. The results in RMSE are shared in our replication package.

warm-start scenario still outperform the ones from cold-start scenarios throughout all commits during development.

Our approach demonstrates the effectiveness of incorporating historical measurements in selector initialization and performance model training, particularly in maintaining consistency and reliability across different commits.

VII. Threats to Validity

External Validity. In our study, we used four subject systems in different domains for the configuration performance prediction in previous works [14], [20]. These systems have an open-source development history based on the Git version control, and execution of the configuration is convenient. However, we cannot ensure our results cover all the domains of the software, especially the closed-source and unconfigurable software.

Internal Validity. Even though we have repeated our experiment several times, we can not avoid the randomness influencing our results. In addition, the tuning of the parameter in other sampling approaches may influence the comparison of the approaches.

Construct Validity. In our study, we mainly test the processing time of the subject systems and do not cover other metrics, such as CPU usage or memory usage. We will extend our approach into more scenarios in the future.

VIII. Conclusion

In this paper, we present CoMSA, a modeling-driven sampling approach for configuration performance testing. CoMSA is designed in scenarios where there does not exist historical performance testing results (code start) and when there exists historical information (warm start). Our evaluation results show that CoMSA outperforms other baseline sampling approaches in building performance models with higher accuracy. The results also demonstrate the benefit of leveraging historical performance testing results in the warm-start scenarios. This paper provides the following contribution:

- We propose a sampling approach CoMSA to reduce the cost of configuration performance testing.
- CoMSA can utilize historical performance testing results to further reduce the code of configuration performance testing during software development.

REFERENCES

[1] "Circleid. misconfiguration brings down entire .see domain in sweden." https://circleid.com/posts/misconfiguration_brings_down_entire_se_domain_in_sweden/, accessed: 2023-05-02.

[2] "R. johnson. more details on today's outage." https://rb.gy/yh4c9, accessed: 2023-05-02.

[3] "Relative standard deviation," https://www.chem.tamu.edu/class/fyp/keeney/stddev.pdf, accessed: 2023-01-15.

[4] L. Bao, X. Liu, Z. Xu, and B. Fang, "Autoconfig: automatic configuration tuning for distributed message systems," in *ASE*. ACM, 2018, pp. 29–40.

[5] A. G. Bedeian and K. W. Mossholder, "On the use of the coefficient of variation as a measure of diversity," *Organizational Research Methods*, vol. 3, no. 3, pp. 285–297, 2000.

[6] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "Distribution-aware sampling and weighted model counting for SAT," in *AAAI*. AAAI Press, 2014, pp. 1722–1730.

[7] S. Chen, Y. Liu, I. Gorton, and A. Liu, "Performance prediction of component-based applications," *J. Syst. Softw.*, vol. 74, no. 1, pp. 35–43, 2005.

[8] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*. ACM, 2016, pp. 785–794.

[9] W. Fu and T. Menzies, "Easy over hard: a case study on deep learning," in *ESEC/SIGSOFT FSE*. ACM, 2017, pp. 49–60.

[10] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," *Eng. Appl. Artif. Intell.*, vol. 115, p. 105151, 2022.

[11] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-aware performance prediction: A statistical learning approach," in *ASE*. IEEE, 2013, pp. 301–311.

[12] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2208–2221, 2011.

[13] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, and H. Yu, "Data-efficient performance learning for configurable systems," *Empir. Softw. Eng.*, vol. 23, no. 3, pp. 1826–1867, 2018.

[14] H. Ha and H. Zhang, "Deepperf: performance prediction for configurable software with deep sparse neural network," in *ICSE*. IEEE / ACM, 2019, pp. 1095–1106.

[15] ——, "Performance-influence model for highly configurable software with fourier learning and lasso regression," in *ICSME*. IEEE, 2019, pp. 470–480.

[16] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *ICSE (1)*. IEEE Computer Society, 2015, pp. 517–528.

[17] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[18] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: an exploratory analysis," in *ASE*. IEEE Computer Society, 2017, pp. 497–508.

[19] D. Jin, X. Qu, M. B. Cohen, and B. Robinson, "Configurations everywhere: implications for testing and debugging in practice," in *ICSE Companion*. ACM, 2014, pp. 215–224.

[20] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel, "Distance-based sampling of software configuration spaces," in *ICSE*. IEEE / ACM, 2019, pp. 1084–1094.

[21] H. J. Kang, T. F. Bissyandé, and D. Lo, "Assessing the generalizability of code2vec token embeddings," in *ASE*. IEEE, 2019, pp. 1–12.

[22] E. Krause, *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*, ser. Dover Books on Mathematics Series. Dover Publications, 1986.

[23] R. Krishna, V. Nair, P. Jamshidi, and T. Menzies, "Whence to learn? transferring knowledge in configurable systems using BEETLE," *IEEE Trans. Software Eng.*, vol. 47, no. 12, pp. 2956–2972, 2021.

[24] C. Lemieux, R. Padhye, K. Sen, and D. Song, "Perffuzz: automatically generating pathological inputs," in *ISSTA*. ACM, 2018, pp. 254–265.

[25] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, and S. Sajedi, "Using black-box performance models to detect performance regressions under varying workloads: an empirical study," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 4130–4160, 2020.

[26] M. Lillack, J. Müller, and U. W. Eisenecker, "Improved prediction of non-functional properties in software product lines with domain context," in *Software Engineering*, ser. LNI, vol. P-213. GI, 2013, pp. 185–198.

[27] Q. Luo, D. Poshyvanyk, and M. Grechanik, "Mining performance regression inducing code changes in evolving software," in *MSR*. ACM, 2016, pp. 25–36.

[28] H. Martin, M. Acher, J. A. Pereira, L. Lesoil, J. Jezequel, and D. E. Khelladi, "Transfer learning across variants and versions: The case of linux kernel size," *IEEE Trans. Software Eng.*, vol. 48, no. 11, pp. 4274–4290, 2022.

[29] J. Martinez, J. Sottet, A. Garcia Frey, T. F. Bissyande, T. Ziadi, J. Klein, P. Temple, M. Acher, and Y. L. Traon, "Towards estimating and predicting user perception on software product variants," in *ICSR*, ser. Lecture Notes in Computer Science, vol. 10826. Springer, 2018, pp. 23–40.

[30] S. Mühlbauer, S. Apel, and N. Siegmund, "Identifying software performance changes across variants and versions," in *ASE*. IEEE, 2020, pp. 611–622.

[31] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using FLASH," *IEEE Trans. Software Eng.*, vol. 46, no. 7, pp. 794–811, 2020.

[32] J. Oh, D. S. Batory, M. Myers, and N. Siegmund, "Finding near-optimal configurations in product lines by random sampling," in *ESEC/SIGSOFT FSE*. ACM, 2017, pp. 61–71.

[33] J. A. Pereira, M. Acher, H. Martin, and J. Jezequel, "Sampling effect on performance prediction of configurable systems: A case study," in *ICPE*. ACM, 2020, pp. 277–288.

[34] J. A. Pereira, M. Acher, H. Martin, J. Jezequel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," *J. Syst. Softw.*, vol. 182, p. 111044, 2021.

[35] R. Queiroz, T. Berger, and K. Czarnecki, "Towards predicting feature defects in software product lines," in *FOSD*. ACM, 2016, pp. 58–62.

[36] T. P. Ryan, *Statistical methods for quality improvement*. John Wiley & Sons, 2011.

[37] F. Samreen, Y. El-khatib, M. Rowe, and G. S. Blair, "Daleel: Simplifying cloud instance selection using machine learning," in *NOMS*. IEEE, 2016, pp. 557–563.

[38] M. Sayagh, N. Kerzazi, B. Adams, and F. Petrillo, "Software configuration engineering in practice interviews, survey, and systematic literature review," *IEEE Trans. Software Eng.*, vol. 46, no. 6, pp. 646–673, 2020.

[39] Y. Shu, Y. Sui, H. Zhang, and G. Xu, "Perf-al: Performance prediction for configurable software through adversarial learning," in *ESEM*. ACM, 2020, pp. 16:1–16:11.

[40] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *ICSE*. IEEE Computer Society, 2012, pp. 167–177.

[41] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov, "Scalable prediction of non-functional properties in software product lines," in *SPLC*. IEEE Computer Society, 2011, pp. 160–169.

[42] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Trans. Software Eng.*, vol. 45, no. 7, pp. 683–711, 2019.

[43] W. L. Teoh, M. B. C. Khoo, P. Castagliola, W. C. Yeong, and S. Y. Teh, "Run-sum control charts for monitoring the coefficient of variation," *Eur. J. Oper. Res.*, vol. 257, no. 1, pp. 144–158, 2017.

[44] P. Valov, J. Guo, and K. Czarnecki, "Empirical comparison of regression methods for variability-aware performance prediction," in *SPLC*. ACM, 2015, pp. 186–190.

[45] Y. Xiang, H. Huang, Y. Zhou, S. Li, C. Luo, Q. Lin, M. Li, and X. Yang, "Search-based diverse sampling from real-world software product lines," in *ICSE*. ACM, 2022, pp. 1945–1957.

[46] T. Xu and Y. Zhou, "Systems approaches to tackling configuration errors: A survey," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 70:1–70:41, 2015.

[47] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *SOSP*. ACM, 2011, pp. 159–172.

[48] S. Zhang and M. D. Ernst, "Which configuration option should I change?" in *ICSE*. ACM, 2014, pp. 152–163.