Bachelor's Thesis

Time-Inhomogeneous Performance Modeling

Sude Ergün

September 28, 2025

Advisors: Tobias Dick
Kallistos Weis

Reviewers: Prof. Dr.-Ing. Sven Apel
Prof. Dr.-Ing. Martina Maggio

UNIVERSITÄT
DES
SAARLANDES

Chair of Software Engineering ◇ Saarland Informatics Campus ◇ Saarland University

Erklärung Statement

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne die Beteiligung dritter Personen verfasst habe, und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht. Insbesondere bestätige ich hiermit, dass ich bei der Erstellung der nachfolgenden Arbeit mittels künstlicher Intelligenz betriebene Software (z. B. ChatGPT) ausschließlich zur Textüberarbeitung/-korrektur und zur Code-Vervollständigung und nicht zur Bearbeitung der in der Arbeit aufgeworfenen Fragestellungen zu Hilfe genommen habe. Alle mittels künstlicher Intelligenz betriebenen Software (z. B. ChatGPT) generierten und/oder bearbeiteten Teile der Arbeit wurden kenntlich gemacht und als Hilfsmittel angegeben. Ich erkläre mich damit einverstanden, dass die Arbeit mittels eines Plagiatsprogrammes auf die Nutzung einer solchen Software überprüft wird. Mir ist bewusst, dass der Verstoß gegen diese Versicherung zum Nichtbestehen der Prüfung bis hin zum Verlust des Prüfungsanspruchs führen kann.

I hereby declare that I have written this thesis independently and without the involvement of third parties, and that I have used no sources or aids other than those indicated. All passages taken directly or indirectly from publications or other external sources have been identified as such. In particular, I confirm that I have used AI-based software (e.g., ChatGPT) exclusively for the following permitted sub-tasks: text rewriting/revision and code completion, and not to address or formulate the main research questions of the thesis. All parts of the thesis that were generated and/or edited using AI-based software (e.g., ChatGPT) have been disclosed and documented in accordance with the documentation requirements. I agree that the thesis may be checked using plagiarism detection software, including checks for the use of such software. I am aware that any violation of this declaration may result in failing the examination and lead to losing the right to be examined.

Saarbrücken, 29.09.25 S. Ergun
(Datum Date) (Unterschrift Signature)

Einverständniserklärung (optional) Declaration of Consent (optional)

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

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

Saarbrücken, 29.09.25 S. Ergun
(Datum Date) (Unterschrift Signature)

Abstract

System performance changes throughout execution, as properties such as memory consumption or latency vary due to changing workloads and internal states. This temporal behavior can lead to bottlenecks and performance peaks. Even worse, in configurable systems, performance varies not only over time but also across configurations, since different configuration options can affect performance significantly. To support performance analysis of configurable systems, performance influence models are used to capture how individual configuration options and their interactions influence the performance of a system. Existing performance influence models predict only scalar performance values, such as the minimum, maximum, or average value for the whole program execution. As a result, we have no knowledge about how the performance varies over the execution time. Understanding temporal behavior would allow a more detailed analysis of performance and help identify when resource limits are reached or slowdowns occur, enabling performance improvements.

In this work, we extend performance models to predict a sequence describing how the performance property in question varies throughout execution. To achieve this, we apply the Discrete Fourier Transform to represent time series in the frequency domain, construct a model that captures how configuration options and their interactions influence the frequency values, and transform the performance values back into the time domain using the Inverse Discrete Fourier Transform.

We evaluate our approach in a controlled setting using seeded performance behavior in real-world case studies. This allows us to assess the accuracy of our model's predictions using performance data for which the ground truth is known by design. That means, we are able to compare the predicted performance values with the actual performance values and assess how well the model captures temporal patterns and the impact of configuration options. Overall, by modeling how performance properties vary throughout execution instead of summarizing them as single values, our approach provides a more informative and interpretable representation of system performance.

Acknowledgments

To begin with, I would like to express my deepest appreciation to my advisor, Kallistos Weis, who has been an inspiring mentor not only during this thesis but also long before throughout my academic journey. This work would not have been possible without his constant encouragement, extensive and insightful feedback, and willingness to dedicate his time so generously. I truly enjoyed working with him and look forward to continuing this journey.

Furthermore, I would like to thank my advisor Tobias Dick for his constructive feedback and valuable contributions to this work. I deeply appreciate his openness and willingness to always lend an ear.

My sincere thanks go to Professor Sven Apel for supervising this work and always having time to provide thoughtful comments and indispensable recommendations. I am grateful to Professor Martina Maggio for her guidance, invaluable input, and expert assistance with figure preparation, often accompanied by the pleasure of sharing good tea.

I would like to wholeheartedly thank my family and friends for their unwavering support throughout my life and during this work. With love and gratitude, I thank Aeneas Sommer, whose constant support and encouragement, especially during the most challenging phases, made everything more manageable.

Contents

List of Figures

List of Tables

1 Introduction

Almost every complex software system today is highly configurable [41]. This variability introduces a vast configuration space, rendering it infeasible to evaluate the performance of all configurations [15, 30]. Different configurations frequently result in different performance behavior, such as variations in latency, memory usage or throughput [45]. For example, in a database system, enabling caching can reduce latency, while increasing the number of threads typically improves throughput but might also raise memory consumption. In order to predict the performance behavior of a specific configuration, developers need ways to reason about the influence of different configuration options and their interactions on performance, since the overall performance arises from their combined contribution [4, 27, 37, 39]. Therefore, a major challenge lies in understanding how individual configuration options and their interactions influence the overall performance. Quantifying these influences not only supports understanding, but also facilitates debugging and optimization by helping to localize performance issues within the configuration space [43]. This enables targeted improvements to specific configuration options or interactions that impact performance [43].

Previous work aimed to understand the relationship between configuration options and performance by learning performance influence models [35, 36]. A performance influence model describes the influence of configuration options and their interactions on the performance of a system. Siegmund et al. [35] demonstrated that it is feasible to infer such performance influence models with high accuracy in predicting performance values of configurations. In performance influence models, the performance of a configuration is represented as a scalar value [35, 36]. These scalar values are commonly used to capture total execution time or aggregate values, such as total energy consumption, or the minimum, maximum, or average of non-functional properties like latency or throughput. However, when considering, for example, memory consumption, relying only on the minimum or average ignores potential short-term peaks, which might lead to memory shortages and reduced system reliability. On the other hand, relying solely on the maximum would assume worst-case behavior for the entire execution, which overestimates the systems resource consumption and therefore leads to inefficient resource allocation. Consequently, summarizing performance as a single scalar value can misrepresent the actual performance behavior of a system, leading either to insufficient or excessive resource provisioning. Moreover, it results in unrealistic assumptions, as many performance properties vary over time. In fact, in many cases the value of the performance property varies during execution, e.g., software applications tend to allocate an increasing amount of memory over time.

To capture these performance variations, we aim to predict not just a scalar performance value but an entire sequence of performance values across different points in time during execution. In this thesis, we present an approach to predict a sequence that captures how the

performance of a system varies over execution time for a given configuration. To achieve this, we use the Discrete Fourier Transform to transform a time series of performance values into the frequency domain. We implement this approach as a framework that takes a configuration as input and returns a predicted performance sequence over time.

The advantage of our approach is that it is not required to learn a model that directly predicts the entire performance sequence over time. In addition, it is not necessary to predict a sequence of time-based performance values, which would require more complex sequence models. Instead, we make use of the frequency representation of our measurements. The frequency representation is more compact and reveals structural patterns. It allows us to express the performance behavior over time as a set of scalar frequency values. As a result, we reduce the original problem of predicting a sequence to the familiar problem of predicting scalar values. This allows us to apply existing modeling techniques to the frequency values. Consequently, we learn a model, which describes how configuration options and their interactions influence the values for the frequencies.

To learn the model, we first create a sample set of configurations, which is a subset of the entire configuration space, using random sampling. For each configuration in the sample set, we measure the performance property of interest at equal time intervals during execution to get a time series of performance measurements. Next, we transform each of the time series into the frequency domain using the Discrete Fourier Transform. This allows us to formulate an optimization problem for the frequencies, which we solve using the optimization solver Gurobi. To predict the performance sequence for an unseen configuration, we give it as input to the model, which predicts the frequency representation of the desired performance sequence. Finally, by applying the Inverse Discrete Fourier Transform on these frequency values, we can reconstruct the full temporal behavior of the performance property.

To evaluate our approach in a controlled setting, we design a set of seeded case studies with predefined performance functions for each configuration. This allows us to establish a ground truth for the performance functions, which we can use in the measurement process and to evaluate the accuracy of our approach. We construct the case studies by taking the configuration space of real-world configurable software systems and manually seeding functions for configuration options and their interactions, which determine their influence on performance over time. The performance function for a configuration is computed as the sum of the influences of all configuration options and interactions involved in the configuration. We vary both the number and choice of configuration options assigned influences, as well as the type of influence functions. Models are constructed using three different sample sizes to examine how the number of sampled configurations affects prediction accuracy. To evaluate the applicability of our approach in real-world scenarios, we assess the impact of synthetic noise on prediction accuracy by considering measurements both with and without added noise. For the noisy measurements, we add random Gaussian noise to each point in the performance sequence of every configuration.

Using these seeded case studies, we generate predicted sequences for all configurations in the configuration space and analyze the predictions over time for a selected subset of configurations, comparing them to their actual sequences. This analysis is performed for multiple configurations across all sample sizes, taking into account whether a configuration was included in the sample set. To capture the overall prediction accuracy across the entire

configuration space, we summarize the difference between the predicted and actual sequences using the mean Integral Squared Error.

We show that our approach to build time-inhomogeneous performance models is capable of constructing models with high prediction accuracy. In our experiment, we observe that prediction accuracy improves with increasing sample size, with models built with larger samples exactly matching the actual performance sequences. Models constructed with smaller sample sizes lose some accuracy but generally capture the overall trends of the performance behavior. When synthetic noise is added, models constructed with small sample sizes tend to overfit, interpreting noise as part of the underlying behavior, whereas models constructed with larger sample sizes better regularize the noise and maintain highly accurate predictions.

In summary, we make the following contributions in this thesis:

- We present a novel approach to predicting performance sequences over execution time for a given configuration of a configurable software system.
- We reduce the problem of learning time dependent performance sequences to predicting their frequency representation by applying the Discrete Fourier Transform.
- We formulate an optimization problem that assigns influences of configuration options and their interactions to the measured frequency values, resulting in a model that predicts the performance behavior of unseen configurations in the frequency domain.
- We provide a Gurobi-based prototype implementation of our approach that constructs a time-inhomogeneous performance model, which takes a configuration as input and returns the predicted performance sequence over time.
- We design seeded case studies with known performance behavior over time to serve as benchmarks. We evaluate our framework using these seeded case studies to assess its accuracy in predicting the performance sequence over time.

2

Background

In this section, we provide the necessary background to understand our approach. We first introduce the concept of configurable software systems, including configuration options, configurations, and configuration spaces. Then, we introduce the concept of performance measurements paired with the concept of time series.

2.1 Configurable Software Systems

A common approach to model the variability of a software system is to use a configuration options [20]. These configuration options are distinct functionalities and capabilities that a software system provides to users in order to fulfill specific requirements [2]. They also express commonalities and variability among software variants [2]. We denote the set of all configuration options of a software system as $\mathcal{O} = \{o_1, \dots, o_{n_o}\}$. Configuration options can be either enabled or disabled, which means the corresponding functionality is either included in the software system or not, respectively. We abstract this by defining configuration options as boolean variables, where an enabled configuration option is represented by the value 1 and a disabled configuration option by the value 0. Assigning a value (i. e., 1 or 0) to each configuration option of the software system results in a *configuration* c. A configuration is a mapping from the set of configuration options to their values, i. e., $c : \mathcal{O} \to \{1, 0\} \equiv \{(o_1, x_1), \dots, (o_{n_o}, x_{n_o})\}$. Therefore, it holds that $x_i \in \{1, 0\}$ is the value assigned to configuration option o_i in configuration c. The set of all possible configurations is called the *configuration space* and is denoted by $\mathcal{C} = \{c \mid c : \mathcal{O} \to \{1, 0\}\}$.

In configurable software systems, not every configuration is valid, since there are often *constraints* on configuration options to capture dependencies and restrictions between them, such as requiring that certain configuration options must be enabled together or cannot be enabled at the same time. We model these constraints using logical expressions, like implications or mutual exclusion. We define the set of all valid configurations as $\mathcal{V} \subseteq \mathcal{C}$, where each $c \in \mathcal{V}$ satisfies the configuration constraints of the system.

In addition to individual configuration options, we also consider subsets of configuration options that occur together, which we call *configuration option interactions*. A configuration option interaction τ is a set of configuration options, i. e., $\tau \subseteq \mathcal{O}$. It can consist of a single option, e. g., $\tau = \{o_1\}$, or multiple options, e. g., $\tau = \{o_1, o_2\}$. We say that a configuration option interaction τ is active (or present) in a configuration c if all options in τ are enabled in

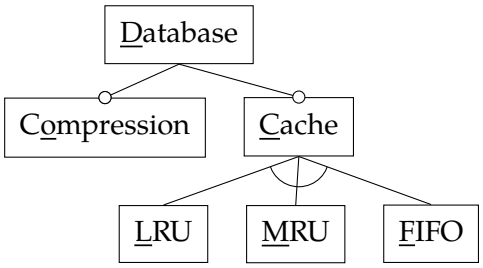


Figure 2.1: Feature diagram for the Database system.

c. To check whether a configuration c satisfies all conditions specified by an interaction τ, we define an indicator function:

$$\text{on}(c, \tau) = \begin{cases} 1 & \text{if } \forall o \in \tau : c(o) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Example 1 (Database System)**.** Consider a simple Database system with the set of configuration options $\mathcal{O} = \{d, o, c, l, m, f\}$, where Database represents the base functionality of the Database system. A user can optionally enable Compression to reduce the size of the data stored, and is able to choose between different Cache eviction policies, i. e., LRU, MRU, and Fifo. Figure 2.1 shows the feature diagram of the Database system, representing all possible configuration options with their respective constraints. This results in the set of valid configurations $\mathcal{V} = \{d, do, dcl, dcm, dcf, docl, docm, docf\}$. Note that, to simplify the notation, we refer to a configuration by a string of configuration options, where we only include the enabled configuration options. For example, the configuration $c_1 = dcf$ represents an instance of the Database system with a first-in first-out cache eviction policy, while the configuration $c_2 = dcl$ represents the database with a least-recently-used cache eviction policy. In this context, the configuration option interaction $\tau = \{c, f\}$ is active in configuration c_1, since $\text{on}(c_1, \tau) = 1$, but inactive in configuration c_2, where $\text{on}(c_2, \tau) = 0$.

2.2 Performance Measurements

The performance of a configurable software system depends on the configuration of the system [45]. There are different quantitative properties that can be measured to capture the performance of a software system, such as execution time, energy consumption, memory usage, or throughput. Such measurable quantities, referred to as *performance metrics*, describe (non-functional) aspects of the execution of a (configurable) software system.

We distinguish between performance metrics that are *time-homogeneous* and *time-inhomogeneous*. Time-homogeneous performance metrics are measures that can be expressed as a single value, such as the total execution time, total energy consumption, or average memory usage of a software system. Time-inhomogeneous performance metrics are measures that vary over time, such as the memory usage or the energy consumption of a software system during its execution.

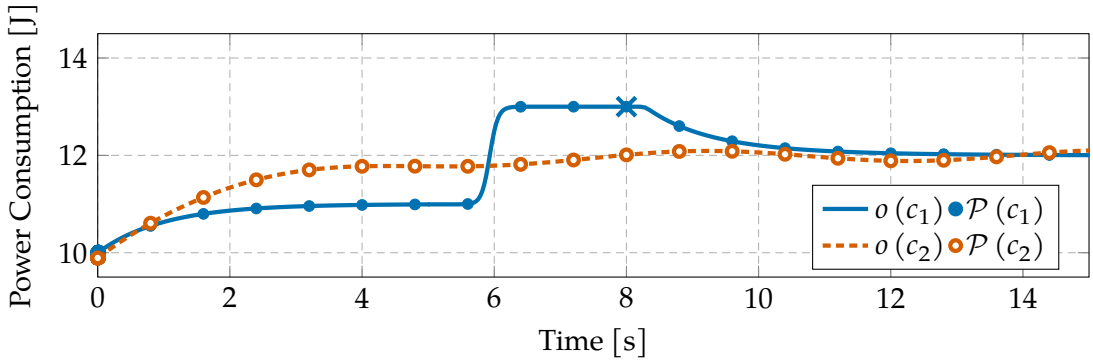


Figure 2.2: The blue (—) and red lines (- -) represent the actual function of the power consumption as time-inhomogeneous performance metric for the DATABASE system under configurations c_1 and c_2. The corresponding blue (●) and red dots (○) indicate concrete performance measurements taken at discrete time steps during execution under the respective configurations.

Definition 1 (Performance Measurement)**.** A *performance function* $\texttt{perf}_{\texttt{metric},c} : \mathbb{R}_{\geq} \to \mathbb{R}$ measures the performance metric for a configuration c at a given time t. We denote the performance function $\texttt{perf}_{\texttt{metric},c}$ for a specific performance metric with m_c. Then, a *performance measurement* for configuration c at time t is the value $m_c(t)$[1].

Example 2 (Performance Measurement)**.** Continuing our example of the DATABASE system, we chose power consumption $[J]$ as our performance metric of interest. In Figure 2.2, we show the actual function of the power consumption for the DATABASE system under configurations c_1 (—) and c_2 (- - ·), respectively. The (●, ○) represent the concrete performance measurements taken at discrete time steps during the execution of the DATABASE system under the respective configurations. Each dot represents the performance measurement $(m_{c_1}(t))$ or $(m_{c_2}(t))$ at a specific point in time t for the power consumption of the system under its corresponding configuration c_1 or c_2, respectively. For example, × represents the power consumption (13 J) of the DATABASE system under configuration c_1 at time $t = 8\,\text{s}$.

For a time-inhomogeneous performance metric and a configuration c, we collect a sequence of performance measurements starting at a point in time $t_{\texttt{ini}}$ with the sampling period $t_{\texttt{sam}}$. We measure the performance metric at discrete time steps $t_k = t_{\texttt{ini}} + k \cdot t_{\texttt{sam}} \in \mathbb{R}_{\geq}$, where $k \in \mathbb{N}$ is the index of the measurement. This results in a sequence of performance measurements $(m_c(t_k))_{k\in\{0,\dots,n_p-1\}}$, where n_p is the number of measurements.

Definition 2 (Performance Trace)**.** A *performance trace* for a configuration c is a tuple $p = \langle t_{\texttt{ini}}, t_{\texttt{sam}}, (m_{c,k})_{k\in\{0,\dots,n_p-1\}}\rangle$, where $(m_{c,k})_{k\in\{0,\dots,n_p-1\}}$ is a sequence of performance measurements, i. e., $(m_{c,k})_{k\in\{0,\dots,n_p-1\}} \equiv (m_c(t_k))_{k\in\{0,\dots,n_p-1\}}$.

The performance data for each configuration of the software system is represented as such a performance trace. Since performance behavior potentially varies across configurations, the corresponding performance traces may also differ, which means, we have to collect performance traces for multiple configurations.

1 For a time-homogeneous performance metric, we can omit the time component and simply write m for the single performance value.

Definition 3 (Performance Dataset)**.** A *performance dataset* is a set of performance traces $\mathcal{P} = \{\langle c_\ell, p_\ell \rangle_{\ell \in \{0,\ldots,n_c-1\}} \mid c_\ell \in \mathcal{C}\}$, where each performance trace p_ℓ corresponds to a different configuration c_ℓ of the software system and n_c is the number of measured configurations.

Example 3 (Performance Dataset)**.** The sequences $(\bullet_k)_{k\in\{0,\ldots,18\}}$ and $(\circ_k)_{k\in\{0,\ldots,18\}}$ (c.f. Figure 2.2) represent the sequences of performance measurements for the configurations c_1 and c_2, respectively. Each sequence, together with $t_{\texttt{ini}} = 0$ and $t_{\texttt{sam}} = 0.8$, results in a performance trace, i. e., $p_1 = \langle 0, 0.8, (\bullet_k)_{k\in\{0,\ldots,18\}} \rangle$ and $p_2 = \langle 0, 0.8, (\circ_k)_{k\in\{0,\ldots,18\}} \rangle$. Combining these performance traces for both configurations results in the performance dataset $\mathcal{P} = \{\langle c_1, p_1 \rangle, \langle c_2, p_2 \rangle\}$.

3

Approach

Performance modeling is a well-established research area within software engineering that focuses on understanding and predicting the performance of software systems [35]. The goal is to provide insights into how software systems behave under different configurations, enabling developers and users to make informed decisions about system design and configuration. The literature on performance modeling focuses on explaining and predicting *a posteriori* performance properties [35, 45], i. e., on modeling the performance measured at the end of the execution of the software system. A classical example is the modeling of the execution time or throughput of a software system in a particular configuration [35, 45].

In this thesis, we extend the classical performance modeling approach with a new dimension: the *time dimension*. These models are referred to as *time-inhomogeneous performance models*. The time dimension allows us to model how performance metrics behave over time, rather than just at the end of the execution. This is particularly relevant for software systems and performance metrics that exhibit time-dependent behavior, such as memory usage of a database system, latency of a web service, or energy consumption of an embedded system.

A common approach to analyze time-dependent behavior is to use the Discrete Fourier Transform [5, 26, 34, 38]. The Discrete Fourier Transform finds application wherever analyzing frequency representations or identifying hidden structure in data is beneficial [24, 28, 32]. It converts a discrete sequence of samples from a signal into its frequency representation. The input of the Discrete Fourier Transform is a finite sequence of real or complex numbers. The sequence needs to be equally spaced and typically represents samples of a signal. The Discrete Fourier Transform converts the signal from its original domain, which is often the time (or space domain), to a representation in the frequency domain. This function maps the input sequence (i. e., our measurement sequence $(m_{c_\ell,k})$) to a sequence of complex numbers, where each complex number corresponds to its respective frequency component. More concretely, the k-th complex number represents the contribution and respective strength of the k-th frequency component. In the following we give the definition for complex numbers and the Discrete Fourier Transform tailored to our application domain.

Definition 4 (Discrete Fourier Transform)**.** The *Discrete Fourier Transform* is a function $\mathrm{dft} : \mathbb{R}^{n_p} \to \mathbb{C}^{n_p}$ that maps a finite sequence of n_p measurements of a performance trace to their respective frequency representation, i. e., $\{m_0, \dots m_{n_p-1}\} \mapsto \{\varrho_0, \dots, \varrho_{n_p-1}\}$. The Discrete Fourier Transform is defined as follows:

$$\varrho_k = [\Re(\varrho) + \Im(\varrho)\mathrm{i}]_k := \sum_{j=0}^{n_p-1} m_j \, e^{-2\pi \mathrm{i} \frac{k}{n_p} j}, \quad \text{for } k \in \{0, \dots, n_p - 1\}, \tag{3.1}$$

where i is the imaginary unit and ϱ_k is the coefficient of the frequency component $\omega_k = \frac{k}{n_p} \cdot \frac{1}{t_{\mathsf{sam}}}$.

In order to convert the frequency representation back to the time domain, we can use the Inverse Discrete Fourier Transform $\mathrm{dft}^{-1} : \mathbb{C}^{n_p} \to \mathbb{R}^{n_p}$, which is the mathematical inverse function of the Discrete Fourier Transform.

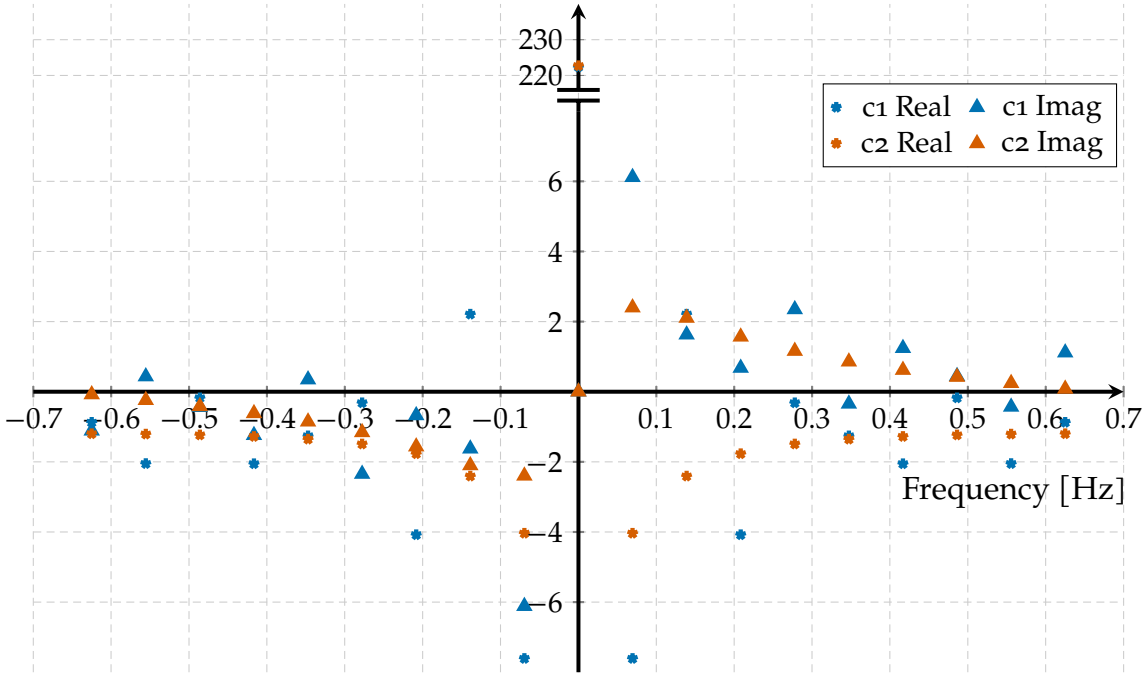


Figure 3.1: By applying the Discrete Fourier Transform to the performance measurement sequences shown as blue (●) and red (●) dots for configurations c_1 and c_2 in Figure 2.2 of the DATABASE example, we obtain the blue (▲) and red (▲) triangles representing the real parts, and blue (✱) and red (✱) stars representing the imaginary parts corresponding to the frequencies shown on the y-axis.

Example 4 (Discrete Fourier Transform)**.** We transform the performance measurement sequences $(●_k)_{k\in\{0,\dots,18\}}$ and $(●_k)_{k\in\{0,\dots,18\}}$ (c.f. Figure 2.2) for the configurations c_1 and c_2 into the frequency domain using the Discrete Fourier Transform. Each measurement sequence yields a corresponding sequence of complex coefficients $(✱+▲i)_{k\in\{0,\dots,18\}}$ and $(✱+▲i)_{k\in\{0,\dots,18\}}$, where the k-th coefficient represents the contribution to the k-th frequency, as illustrated on the x-axis in Figure 3.1. Applying the Inverse Discrete Fourier Transform to these sequences $(✱ + ▲i)_{k\in\{0,\dots,18\}}$ and $(✱ + ▲i)_{k\in\{0,\dots,18\}}$, reconstructs the original measurement sequences $(●_k)_{k\in\{0,\dots,18\}}$ and $(●_k)_{k\in\{0,\dots,18\}}$.

3.1 Model Construction

Our approach leverages frequency domain analysis to learn performance models which can predict for every configuration a performance sequence over time. In what follows, we present the steps of our time-inhomogeneous performance modeling approach, illustrated in Figure 3.2, which builds a framework that takes a configuration as input and returns its predicted performance sequence over time.

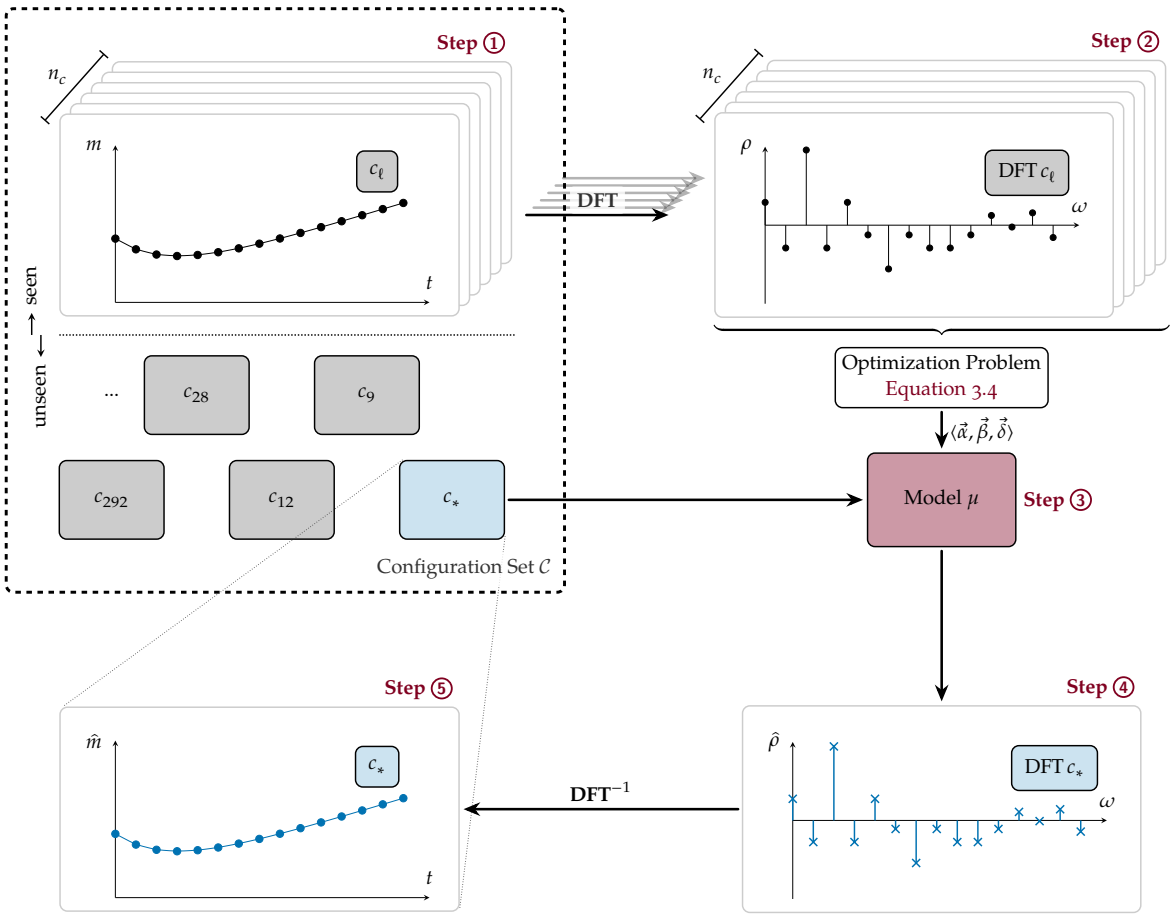


Figure 3.2: Schematic overview of time-inhomogeneous performance modeling.

Step ① To construct a time-inhomogeneous performance model, we first collect performance sequences for a subset of configurations sampled from the configuration set $\mathcal{C}$ of the software system. We generate a sample set of n_c configurations using uniform random sampling, resulting in a finite set of sample configurations $\mathcal{S} = \{c_0, \ldots, c_{n_c-1}\}$.

For each configuration $c_\ell \in \mathcal{S}$, we obtain a performance trace p_ℓ using a common starting time $t_{\texttt{ini}}$, sampling period $t_{\texttt{sam}}$, and number of performance measurements n_p. This guarantees that all sample configurations share the same execution time and are measured at identical time points $\{t_0, \ldots, t_{n_p-1}\}$. Each configuration $c_\ell \in \mathcal{S}$, then yields a sequence of performance measurements $(m_{c_\ell,k})_{k\in\{0,\ldots,n_p-1\}}$, which defines the performance trace $p_\ell = \langle t_{\texttt{ini}}, t_{\texttt{sam}}, (m_{c_\ell,k})_{k\in\{0,\ldots,n_p-1\}} \rangle$. Collectively, these traces form a performance dataset $\mathcal{P} = \{\langle c_\ell, p_\ell \rangle_{\ell\in\{0,\ldots,n_c-1\}} \mid c_\ell \in \mathcal{S}\}$.

Step ② We transform the time series data into the frequency domain using the Discrete Fourier Transform. This transformation allows us to analyze the individual frequency components of the time series data, which enables us to identify patterns and trends in the data.

For all configurations $c_\ell \in \mathcal{S}$, we apply the Discrete Fourier Transform to their measurement sequence $(m_{c_\ell,k})_{k\in\{0,\ldots,n_p-1\}}$ of the corresponding performance trace p_ℓ, yielding the frequency

representation $(\varrho_{c_\ell,k})_{k\in\{0,\dots,n_p-1\}}$. Since all configurations share the same sampling rate $t_{\texttt{sam}}$ and initial measurement time $t_{\texttt{ini}}$, the resulting coefficients $(\varrho_{c_\ell,k})_{k\in\{0,\dots,n_p-1\}}$ are aligned to the same set of frequencies ω_k for $k \in \{0,\dots,n_p-1\}$. While the frequencies are identical across all configurations, the complex-valued coefficients $(\varrho_{c_\ell,k})_{k\in\{0,\dots,n_p-1\}}$ might differ depending on the configuration. In summary, we convert the sample set from sequences of performance measurements to their corresponding frequency representations.

Step ③ We use the frequency components to compute a time-inhomogeneous performance model. This model is an optimal solution to an optimization problem that minimizes the difference between the measured performance data and the model's predictions, formulated in Equation 3.4. The time-inhomogeneous performance model can then be used to predict the performance of the software system in different configurations in the frequency domain. The model can be expressed as a function $\mu : \mathcal{C} \to \mathbb{C}^{|n_p|}$ which maps configurations c_ℓ to a vector whose k-th entry corresponds to the predicted complex coefficient $\varrho_{c_\ell,k}$. The construction of the optimization problem to derive the model is described in detail in the following section. Given the constructed model, we can provide an arbitrary configuration $c_* \in \mathcal{V}$ as input to predict its performance behavior during execution. This includes configurations that were not part of the sample set $\mathcal{S}$ used for model construction. Applying the learned model μ to the configuration c_* yields the predicted frequency representation $\mu(c_*)$ of its performance function.

Step ④ The model outputs the predicted frequency representation of the performance function of c_* by summing, for the k-th entry, the contributions to the frequency value of ω_k from all configuration options and interactions active in c_*.

Step ⑤ Finally, we apply the Inverse Fourier Transform to convert the predictions back to the time domain, allowing us to analyze the performance of the software system over time. We obtain the performance values at the time points $\{t_0,\dots,t_{n_p-1}\}$.

3.2 Optimization Problem

Our goal is to compute a model that captures the influence of individual configuation options and their interactions on a given performance metric over the whole execution time. To achieve this, we assume that we can express the individual influences of the configuration options and their interactions as a linear combination of frequency components. That means, given that there is an inherent connection between the configuration options of the software system and its performance, we try to identify the frequency components and their strength for each configuration option and their interaction that best explain the measured performance data. To this end, we formulate the problem as a minimization problem, where we aim to minimize the difference between the measured frequency coefficients and the model's predictions.
For each frequency ω_k, we aim to determine the contribution of each configuration option interaction τ to the complex coefficient $\varrho_k = [\Re(\varrho) + \Im(\varrho)\mathrm{i}]_k$. For this we introduce weights $\alpha_{\tau,\omega_k} \in \mathbb{R}$ and $\beta_{\tau,\omega_k} \in \mathbb{R}$, where α_{τ,ω_k} captures the influence of interaction τ on the real part $\Re(\varrho)$, while β_{τ,ω_k} captures its influence on the imaginary part $\Im(\varrho)$.

We denote the set of configuration option interactions we consider for model construction as $\mathcal{T}$. To build the set $\mathcal{T}$, we include all terms up to a predetermined size, that is, all individual configuration options and interactions up to a certain size. To check whether a configuration c_ℓ satisfies all conditions specified by an interaction $\tau \in \mathcal{T}$, we use the previously defined indicator function $\mathrm{on}(c_\ell, \tau)$, which returns 1 if τ is active in c_ℓ, and 0 otherwise.

Constraints. We apply the Discrete Fourier Transform to the performance sequence $(m_{c_\ell,k})_{k\in\{0,\dots,n_p-1\}}$ of each configuration c_ℓ, yielding the corresponding complex coefficients $(\varrho_{c_\ell,k})_{k\in\{0,\dots,n_p-1\}}$. With these coefficients, we define the constraints in the optimization problem for every measured configuration c_ℓ, frequency ω_k, and a set of interactions $\mathcal{T}$. The constraints are defined as follows:

$$\left\| \sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\alpha_{\tau,\omega_k} - \Re(\varrho_{c_\ell,k}) + \Big(\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\beta_{\tau,\omega_k} - \Im(\varrho_{c_\ell,k})\Big)\mathrm{i} \right\|_2 \le \delta_{c_\ell,\omega_k}, \quad k\in\{0,\dots,n_p-1\},\ \ell\in\{0,\dots,n_c-1\} \tag{3.2}$$

Equivalently, expanding the $\mathcal{L}_2$ norm, we can express the constraints as:

$$\Big(\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\alpha_{\tau,\omega_k} - \Re(\varrho_{c_\ell,k})\Big)^2 + \Big(\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\beta_{\tau,\omega_k} - \Im(\varrho_{c_\ell,k})\Big)^2 \le \delta^2_{c_\ell,\omega_k}, \quad k\in\{0,\dots,n_p-1\},\ \ell\in\{0,\dots,n_c-1\} \tag{3.3}$$

The predicted real part of coefficient $\varrho_{c_\ell,k} = [\Re(\varrho_{c_\ell}) + \Im(\varrho_{c_\ell})\mathrm{i}]_k$ for configuration c_ℓ at frequency ω_k is given by $\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\alpha_{\tau,\omega_k}$, which sums the contributions α_{τ,ω_k} of all active interactions τ in c_ℓ to the real part $\Re(\varrho_{c_\ell})$. Similarly, the term $\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\beta_{\tau,\omega_k}$ gives the prediction for the imaginary part $\Im(\varrho_{c_\ell})$. Together, these terms form the predicted complex coefficient $\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\alpha_{\tau,\omega_k} + (\sum_{\tau\in\mathcal{T}} \mathrm{on}(c_\ell,\tau)\cdot\beta_{\tau,\omega_k})\mathrm{i}$, which should match the actual coefficient $\varrho_{c_\ell,k}$. We aim to minimize the $\mathcal{L}_2$ norm of the difference between the predicted and observed complex coefficient $\varrho_{c_\ell,k}$. In this way, the optimization problem minimizes the magnitude of the difference between the predicted and actual complex coefficient.

The inequality ensures that the deviation between the predicted and actual coefficient for a configuration c_ℓ and frequency ω_k is bounded by a non-negative error variable δ_{c_ℓ,ω_k}. This error variable introduces flexibility in the optimization problem, enabling it to account for potential errors or uncertainties in the predictions. We introduce an error variable δ_{c_ℓ,ω_k} for each configuration $c_\ell \in \mathcal{S}$ and frequency ω_k. The error variables δ_{c_ℓ,ω_k} represent the $\mathcal{L}_2$ norm of the difference between the predicted and observed coefficients for each configuration c_ℓ and frequency ω_k. In other words, δ_{c_ℓ,ω_k} quantifies the prediction error for configuration c_ℓ at frequency ω_k.

Optimization Problem. The formulated constraints ensure that the predicted coefficients for a configuration c_ℓ and frequency ω_k deviate from the actual ones by at most a bounded

error δ_{c_ℓ,ω_k}. What is missing is that we aim to keep the error as small as possible. Using the constraints, introduced above, we formulate the final minimization problem:

$$\begin{aligned} \min \quad & \sum_{\substack{c_\ell \in \mathcal{S}, \\ k \in \{0,\dots,n_p-1\}}} \delta_{c_\ell,\omega_k}, \\ \text{subject to} \quad & \left\| \sum_{\tau \in \mathcal{T}} \mathrm{on}(c_\ell,\tau) \cdot \alpha_{\tau,\omega_k} - \Re(\varrho_{c_\ell,k}) + \left(\sum_{\tau \in \mathcal{T}} \mathrm{on}(c_\ell,\tau) \cdot \beta_{\tau,\omega_k} - \Im(\varrho_{c_\ell,k}) \right) \mathrm{i} \right\|_2 \leq \delta_{c_\ell,\omega_k}, \\ & \qquad\qquad k \in \{0,\dots,n_p-1\},\ \ell \in \{0,\dots,n_c-1\} \end{aligned} \tag{3.4}$$

The solution of this optimization problem serves as a performance model, which makes predictions in the frequency domain. The objective is to minimize the sum of the error variables δ_{c_ℓ,ω_k}, which represent the deviation between the predicted and actual coefficients across all sample configurations and frequencies. The constraints ensure that, for every configuration and every frequency, the difference between predicted and actual coefficients remains within the bound defined by the error variables. By minimizing the total sum of the error variables, the optimization problem seeks to find the best-fitting model that accurately captures the influence of configuration options and their interactions on the performance metric over time.

We obtain a solution to this optimization problem using the solver Gurobi, which determines optimal contributions α_{τ,ω_k}, β_{τ,ω_k} and δ_{c_ℓ,ω_k} for each interaction based on the frequency data from the sample set. The result is a model $\mu : \mathcal{C} \to \mathbb{C}^{|n_p|}$, which maps configurations to a vector of complex numbers $\{\varrho_{c_\ell,0},\dots,\varrho_{c_\ell,n_p-1}\}$, which we obtain by summing up the predicted complex numbers for each interaction present in the configuration. The k-th entry of the vector is obtained by summing up the predicted contributions of all interactions present in the configuration for the corresponding frequency ω_k.

Algorithm 1: (Gurobi-based) Model Learning

Input: $\mathcal{P}(\mathcal{O})$ $\mathcal{T}$, Performance Dataset database
Output: model

```
model = initialize();
foreach ⟨c, p⟩ ∈ P = {⟨c_ℓ, p_ℓ⟩_{ℓ∈{0,...,n_c}}} = {⟨c_ℓ, ⟨t_ini, t_sam, (m_{c_ℓ,k})_{k∈{0,...,n_p−1}}⟩_ℓ⟩_{ℓ∈{0,...,n_c}}}
  do
    _, t_sam, M = _, t_sam, (m_{c_ℓ,k})_{k∈{0,...,n_p−1}} = p;
    (ϱ_{c_ℓ,k}), (ω_k) = dft (M), dft_freqs (M, t_sam);
    foreach ⟨ω, ϱ⟩ ∈ (ω_k), (ϱ_{c_ℓ,k}) do
      foreach τ in T do
        model.addVars (α_{τ,ω}, β_{τ,ω}, δ_{c,ω});
      π_{c,ω,real} = Σ_{τ∈T} on(c, τ) · α_{τ,ω};
      π_{c,ω,imag} = Σ_{τ∈T} on(c, τ) · β_{τ,ω};
      model.addConstraint (‖(π_{c,ω,real} − ℜ(ϱ)) + (π_{c,ω,imag} − ℑ(ϱ))i‖_2 ≤ δ_{c,ω});
model.minimize(Σ |model.δ|);
return model
```

Algorithmic Implementation. The model construction process is illustrated in algorithm 1. First, we iterate over all configurations and their corresponding performance traces in the performance dataset. For each configuration, we transform its performance trace into the frequency domain using the Discrete Fourier Transform, yielding the complex coefficients $(\varrho_{c_\ell,k})_{k\in\{0,\dots,n_p-1\}}$ associated with the frequencies $(\omega_k)_{k\in\{0,\dots,n_p-1\}}$. Next, for each frequency ω_k, interaction τ, and configuration c_ℓ, we introduce the variables α_{τ,ω_k}, β_{τ,ω_k}, and δ_{c_ℓ,ω_k}, which represent the contribution of the interaction to the real and imaginary parts of the coefficient and the error variable to account for the prediction error. We then add a constraint for the current configuration and frequency, ensuring that the difference between predicted and actual coefficients is bounded by the error variables. Finally, the total sum of all error variables is minimized. This minimization assigns values to the variables α_{τ,ω_k}, β_{τ,ω_k}, and δ_{c_ℓ,ω_k}, such that the the sum of all error variables is minimized, while all constraints are satisfied. Finally, we return the model with the assigned complex values for the influences of each interaction.

Example 5. To illustrate how the optimization problem is constructed for our Database example, we consider the sample set $\mathcal{S} = \{c_1, c_2\}$ and take the interaction set to include all individual configuration options, therefore, it holds $\mathcal{T} = \mathcal{O}$. In the previous example, we computed the Discrete Fourier Transform of the performance measurement sequences for configurations c_1 and c_2, yielding the complex coefficients $(\bigstar + \blacktriangle i)_{k\in\{0,\dots,18\}}$ and $(\bigstar + \blacktriangle i)_{k\in\{0,\dots,18\}}$. This results in the following constraints:

Configuration c_1 :

$$\begin{aligned}
\left\|(\alpha_{d,\omega_0} + \alpha_{c,\omega_0} + \alpha_{f,\omega_0} - \bigstar_0) + (\beta_{d,\omega_0} + \beta_{c,\omega_0} + \beta_{f,\omega_0} - \blacktriangle_0)\mathrm{i}\right\|_2 &\leq \delta_{c_1,\omega_0}\\
\left\|(\alpha_{d,\omega_1} + \alpha_{c,\omega_1} + \alpha_{f,\omega_1} - \bigstar_1) + (\beta_{d,\omega_1} + \beta_{c,\omega_1} + \beta_{f,\omega_1} - \blacktriangle_1)\mathrm{i}\right\|_2 &\leq \delta_{c_1,\omega_1}\\
&\vdots\\
\left\|(\alpha_{d,\omega_{18}} + \alpha_{c,\omega_{18}} + \alpha_{f,\omega_{18}} - \bigstar_{18}) + (\beta_{d,\omega_{18}} + \beta_{c,\omega_{18}} + \beta_{f,\omega_{18}} - \blacktriangle_{18})\mathrm{i}\right\|_2 &\leq \delta_{c_1,\omega_{18}}
\end{aligned}$$

Configuration c_2 :

$$\begin{aligned}
\left\|(\alpha_{d,\omega_0} + \alpha_{c,\omega_0} + \alpha_{l,\omega_0} - \bigstar_0) + (\beta_{d,\omega_0} + \beta_{c,\omega_0} + \beta_{l,\omega_0} - \blacktriangle_0)\mathrm{i}\right\|_2 &\leq \delta_{c_2,\omega_0}\\
\left\|(\alpha_{d,\omega_1} + \alpha_{c,\omega_1} + \alpha_{l,\omega_1} - \bigstar_1) + (\beta_{d,\omega_1} + \beta_{c,\omega_1} + \beta_{l,\omega_1} - \blacktriangle_1)\mathrm{i}\right\|_2 &\leq \delta_{c_2,\omega_1}\\
&\vdots\\
\left\|(\alpha_{d,\omega_{18}} + \alpha_{c,\omega_{18}} + \alpha_{l,\omega_{18}} - \bigstar_{18}) + (\beta_{d,\omega_{18}} + \beta_{c,\omega_{18}} + \beta_{l,\omega_{18}} - \blacktriangle_{18})\mathrm{i}\right\|_2 &\leq \delta_{c_2,\omega_{18}}
\end{aligned}$$

Finally, the optimization problem minimizes the sum of all error variables:

$$\sum_{i=0}^{18} (\delta_{c_1,\omega_i} + \delta_{c_2,\omega_i})$$

The solution of the optimization problem yields the contributions α_{o,ω_k}, β_{o,ω_k} for each configuration option $o \in \{d, o, c, l, m, f\}$ and frequency ω_k. This results in a model $\mu : \mathcal{C} \to \mathbb{C}^{19}$,

$$c \mapsto \begin{bmatrix} \sum_{o \in \{d,o,c,l,m,f\}} (\alpha_{o,\omega_0} + \beta_{o,\omega_0}\mathrm{i}) \cdot \mathrm{on}(c,o) \\ \sum_{o \in \{d,o,c,l,m,f\}} (\alpha_{o,\omega_1} + \beta_{o,\omega_1}\mathrm{i}) \cdot \mathrm{on}(c,o) \\ \vdots \\ \sum_{o \in \{d,o,c,l,m,f\}} (\alpha_{o,\omega_{18}} + \beta_{o,\omega_{18}}\mathrm{i}) \cdot \mathrm{on}(c,o) \end{bmatrix}$$

The frequency representation of a configuration $c \in \mathcal{V}$, is given by applying the learned model to configuration c, i. e., $\mu(c)$, which computes the predicted vector of frequency components by summing up the contributions of all configuration options enabled in c. Applying the Inverse Discrete Fourier Transform to this vector yields the predicted performance sequence $(\hat{m}_{c,k})_{k \in \{0,\ldots,18\}}$ for configuration c.

4

Methodology

The goal of using time-inhomogeneous performance models is to capture the dynamic behavior of configurable software systems allowing for performance predictions that reflect temporal variations. In this section, we present our methodology to evaluate our time-inhomogeneous performance models. We start by stating our research questions, defining the scope of our evaluation, and end with describing the experimental setup to answer our research questions.

4.1 Research Questions

We evaluate our modeling approach guided by two research questions. First, we assess the accuracy of our time-inhomogeneous performance model using manually seeded performance behaviors to examine whether the model can accurately capture the influence of configuration options on performance over execution time. Second, we investigate the impact of synthetic noise on model accuracy.

Accuracy of Time-Inhomogeneous Models. Classical performance modeling approaches, such as linear regression [35, 36], are widely used in the literature to predict a posteriori performance metrics, e.g., response time or throughput [35, 45]. Previous work has shown that classical performance modeling approaches can capture the influence of configuration options on scalar performance values with high accuracy [35, 36].

We designed our approach to capture variant performance behavior over execution time. We achieve this by leveraging the frequency domain and formulating an optimization problem to learn a performance model by minimizing the prediction error. One quality criterion of interest is the accuracy of the predictions made by our time-inhomogeneous performance models. That is, we are interested in how closely the predicted performance sequence aligns with the measured performance sequence for each configuration.

To enable this evaluation, we require a priori knowledge of the exact temporal patterns of the system's performance behavior. We want to control the influence of configuration options and their interactions on overall performance, including how many configuration options are assigned influences and whether interactions also have an influence. Therefore, we consider the configuration space of a real-world configurable software system and manually seed functions representing the performance influences of individual configuration options and their interactions. Using this setup, we examine whether the time-inhomogeneous performance model accurately captures temporal patterns such as peaks, drops, or increases and decreases in performance. This leads to our first research question, in which we investigate the accuracy

of our time-inhomogeneous performance models in capturing variant performance behavior over the execution time.

RQ1: Prediction Accuracy How well do time-inhomogeneous performance models capture temporal patterns of configuration options and their interactions on performance?

Impact of Synthetic Noise. In real-world performance measurements, various factors such as background processes, hardware variability, and network delays can introduce noise. Understanding the influence of noise on model accuracy supports assessing the applicability of our approach in real-world scenarios. To systematically investigate this effect, we simulate noise in our manually seeded case studies by adding synthetic noise to the performance measurements. In our experiments, we focus on Gaussian noise introduced at the configuration level, meaning that noise is added directly to the performance behavior of each configuration. We evaluate the impact of noise on the accuracy of our time-inhomogeneous performance models by comparing predictions obtained from noisy measurements against those derived from noise-free measurements. This analysis allows us to identify potential limitations of our modeling approach and provides insights into how synthetic noise affects model accuracy. Consequently, our second research question focuses on evaluating the accuracy of the time-inhomogeneous models under synthetic noise conditions.

RQ2: Synthetic Noise Effect What is the impact of synthetic noise on the accuracy of time-inhomogeneous performance models?

4.2 Operationalization

To answer our research questions, we introduce the experimental setup and explain how we operationalize our research questions. We specify how our performance measurements are obtained through manually seeded performance behavior and how we assess the accuracy of our time-inhomogeneous performance models using the Mean Integral Squared Error as error metric.

4.2.1 Seeded Performance Behavior

We evaluate our research questions using seeded performance behavior in real world case studies. Each case study is constructed by considering the configuration space of a real-world configurable software system and manually seeding functions for configuration options and their interactions as influence on the overall performance function. We achieve this by defining a set of functions and randomly assigning them to configuration options and their interactions. We compute the performance function of a configuration over execution time by summing up the influences of all configuration options and interactions enabled in the configuration. This allows us to establish a ground truth for the performance functions of all configurations.

Experimental Setup. We consider two real-world configurable software systems from different

Table 4.1: Overview of the subject systems used in our evaluation, including valid configuration space $|\mathcal{V}|$, configuration options $|\mathcal{O}|$, sample sizes (t1, t2, t3) and seeded terms (Int Deg 1, Int Deg 2). "Int Deg 1" refers to an interaction degree of 1, meaning only individual options have an influence, whereas "Int Deg 2" additionally includes influences from pair-wise interactions. Each tuple of seeded terms specifies a scenario, indicating the number of individual configuration options and interactions assigned an influence.

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{O}\rvert$	Sample Size			Seeded Terms (Int Deg 1, Int Deg 2)		
			t1	t2	t3			
LLVM	1 024	11	11	55	165	(11, 0)	(6, 0)	(2, 0)
						(11, 6)	(6, 6)	(2, 6)
LRZIP	432	19	18	90	178	(19, 0)	(10, 0)	(4, 0)
						(19, 10)	(10, 10)	(4, 10)

application domains, including LLVM, a compiler infrastructure, and lrzip, a compression tool. Table 4.1 summarizes the characteristics of these systems and how we seed performance behavior to configuration options and interactions for our evaluation.

For each system, we define three different sample sizes (t1, t2, t3) of configurations, corresponding to the size of t-wise coverage sample sets with $t \in \{1, 2, 3\}$, respectively. Using the three sample sizes t1, t2, t3, we construct the models μ_{t1}, μ_{t2}, μ_{t3}, respectively. Specifically, a 1-wise sample size ensures that every configuration option is enabled in at least one configuration and disabled in at least one configuration, whenever possible. A 2-wise sample size guarantees that every possible value combination of any two configuration options, e. g., both enabled, both disabled, one enabled and the other disabled, and vice versa, appears in at least one configuration of the sample. Similarly, a 3-wise sample size ensures that all possible value combinations of any three configuration options appear in at least one configuration. The required sample sizes for each system and coverage are shown in Table 4.1.

The *Seeded Terms* columns in Table 4.1 specify how many configuration options and interactions are assigned a performance influence. We consider three scenarios in which either all, half, or one-fifth of the configuration options are assigned an influence. For LLVM this corresponds to eleven, six, and two configuration options, respectively. Each column of *Seeded Terms* represents one of these scenarios with two variants. In the first variant (e. g., (11, 0)), only the selected configuration options themselves are assigned influences. In the second variant (e. g., (11, 6)), the same options with the same influences are considered together with 2-wise interactions having an influence assigned. For all scenarios, the number of assigned interactions is set to half the total number of configuration options in the system. For example, in LLVM this results in six pair-wise interactions in all scenarios. For interactions of configuration options, the selected interactions remain constant across all scenarios and repeated assignments within a system, while only their assigned functions vary (e. g., (11, 6) and (6, 6) both have the same six interactions assigned). This ensures comparability between variants with and without 2-wise interactions. The assigned functions are randomly chosen from a set of predefined functions, which is illustrated in Table 4.2. We include multiple functions in the groups of constant, sine, cosine, polynomial, exponential and piecewise functions. Since we only consider the influence of individual configuration options and interactions of size

Table 4.2: Set of functions used for randomly assigning influences to configuration options and their interactions.

Function Type	Functions	
Constant	$f(t) = 1$	$f(t) = 5$
	$f(t) = 10$	$f(t) = 25$
	$f(t) = 50$	$f(t) = 100$
Sine/Cosine	$f(t) = \sin(t)$	$f(t) = \cos(t)$
	$f(t) = 5 \cdot \sin(t)$	
Polynomial	$f(t) = t$	$f(t) = t^2$
	$f(t) = 6 \cdot \sqrt{\frac{t}{100}} \cdot (1 - \frac{t}{100})$	
Piecewise	$f(t) = \begin{cases} 0, & t < 10, \\ 10, & 10 \leq t \leq 20, \end{cases}$	$f(t) = \begin{cases} \frac{t^2}{15}, & t < 15, \\ 15, & 15 \leq t < 20 \end{cases}$
Mixed	$f(t) = 5 \cdot (1 - \lvert(2 \cdot (\frac{t}{100}) - 1)\rvert)$	$f(t) = 8 \cdot (\frac{t}{100}) \cdot \exp(-\frac{(t-10)^2}{100})$
	$f(t) = 10 \cdot \sin^2(\pi \cdot \frac{t}{100})$	$f(t) = 6 \cdot \sin^2(\pi \cdot \frac{t}{100}) + 3 \cdot \sin^4(3 \cdot \pi \cdot \frac{t}{100})$

two, we limit influence assignments during model building to these terms. Accordingly, we predefine the set of configuration options and interactions $\mathcal{T}$ to include all individual options and their pair-wise interactions.

Repeated Measurements. For each scenario, i. e., each tuple of assignments for interaction degree 1 and 2 in the seeded terms, we generate two independent assignments for configuration options and interactions, together with their associated functions by using two different seeds. Within a scenario and seed, the variant without interaction influences is subset of the variant with interaction influences as the same configuration options receive identical influences, while the variant with interaction influences includes additional pair-wise influences. For each assignment, we generate performance measurements for all valid configurations. Subsequently, for each of those performance datasets we learn a time-inhomogeneous performance model. This procedure results in $2 \times 3 \times 2 \times 6$ performance datasets to learn corresponding models for the noiseless setting, as we consider 2 subject systems, 3 sample sizes, and 2 assignments for 6 seeded term scenarios.

Sampling Strategy. To construct the sample set of measured configurations, we use uniform random sampling. Since this thesis does not focus on analyzing the effect of different sampling strategies on our approach, we make use of uniform random sampling as a baseline and generate only one fixed sample set for each sample size and system.

Adding Synthetic Noise. To study the effect of synthetic noise, we repeat the above procedure while adding Gaussian noise. We compute the performance function for a configuration by summing the influences of all active configuration options and interactions. Afterwards, we

perturb the resulting performance sequence by adding, to each value, a random number drawn independently from a Gaussian distribution with mean 0 and standard deviation 1.0.

When building the noisy performance datasets, we use the same configuration options, interactions and assigned functions as in the noiseless datasets, allowing us to investigate the isolated influence of synthetic noise on model accuracy. Consequently, the total number of performance datasets to learn corresponding models for is $2 \times 3 \times 2 \times 6 \times 2$, where the last factor of 2 accounts for both the noiseless and noisy settings.

Performance Measurements. To obtain the performance traces for the sampled configurations, we use the performance functions defined by the seeded performance behavior. Since the performance behavior is specified by the experiment design, we can determine the sequence of performance measurements of any configuration by evaluating the corresponding performance function at different points in time. Consequently, all performance sequences are derived from known performance functions, which serve as ground truth. Additionally, we restrict the measurements by using the same start time $t_{\texttt{ini}} = 0$, same sampling period $t_{\texttt{sam}} = 0.2$, and end time $t_{\texttt{fin}} = 20$ for all configurations. This guarantees that all configurations are measured at identical time points and have the same execution time.

4.2.2 Research Questions

We evaluate how accurately the predicted performance behavior matches the actual performance behavior by answering **RQ1** and **RQ2**. For this purpose, we consider the time-inhomogeneous performance models obtained using the methodology described above. For each learned model, we compare the predicted performance sequence against the actual performance sequence for all configurations of the system. This allows us to assess how well each model captured the temporal behavior across the entire configuration space. To quantify the accuracy of our predictions, we use the following error metric.

Mean Integral Squared Error. To capture the average pointwise deviation between the predicted and actual performance measurements of a configuration c, we compute the mean Integral Squared Error.

$$\text{mean (ISE)}(m_c, \hat{m}_c) = \frac{1}{t_{\texttt{fin}} - t_{\texttt{ini}}} \sum_{i=t_{\texttt{ini}}}^{t_{\texttt{fin}}} (m_c(t_i) - \hat{m}_c(t_i))^2 \tag{4.1}$$

The summation over discrete measurement points t_i approximates the integral of the squared error. Thus, the mean integral squared error is a discrete integral of the squared error, normalized by the total execution time $t_{\texttt{fin}} - t_{\texttt{ini}}$.

This metric averages the squared difference between the predicted values $\hat{m}_c$ and the actual performance measurements m_c at each discrete time point t_i. Here, t_i denotes the i-th discrete measurement time point, starting with the initial point $t_{\texttt{ini}}$ and ending with the final point $t_{\texttt{fin}}$. A small mean integral squared error indicates that the predicted performance function closely matches the actual function across all measured time points, while larger deviations at any time point have a higher impact compared to smaller deviations. This metric ensures that large deviations at individual time points do not dominate the evaluation.

5 Evaluation

In this chapter, we present the results of our experiments, examining how well our approach predicts performance behavior over execution time across the different sample sizes. We first report the results for models constructed with measurements without noise and then with noise, and discuss their implications with respect to our research questions. Within each research question, we distinguish between models learned without influences of interactions of configuration options and those that incorporate them. We analyze the predicted sequences in the time domain for individual, representative configurations. To capture prediction errors across all configurations, we summarize the differences between predicted and actual sequences using the mean Integral Squared Error, illustrated with boxplots to show their overall distribution. Finally, we discuss the results, answer our research questions, and address potential threats to validity of our approach.

5.1 Results

We evaluate our models by comparing the predicted performance values over execution time with the actual measurements. We focus on scenarios where half of the configuration options are assigned influences, both with and without additional pairwise influences. For these scenarios, we construct models for all three sample sizes and report results using a fixed seed to select configuration options, pairwise interactions, and their influence functions. Then, we exemplarily analyze configurations that were included in the sample set for all models, excluded only for model μ_{t1}, excluded for both models μ_{t1} and μ_{t2}, and excluded for all models. In every Figure in this section, the left plot shows a corresponding configuration of LLVM, the right plot one from lrzip, with the black line marking actual performance and colored markers showing predictions for the different models. Since the performance behavior is seeded, we omit the y-label for the performance metric. The seeded performance behavior serves as a placeholder for arbitrary performance metrics, such as power consumption, latency, or throughput.

5.1.1 RQ1: Prediction Accuracy

In order to answer the first research question, we aim to assess how well our approach captures the temporal patterns of configuration options and their interactions. To this end, we examine the predictions in the time domain, after applying the Inverse Discrete Fourier Transform to the predicted frequency values. In the following, we distinguish between performance data

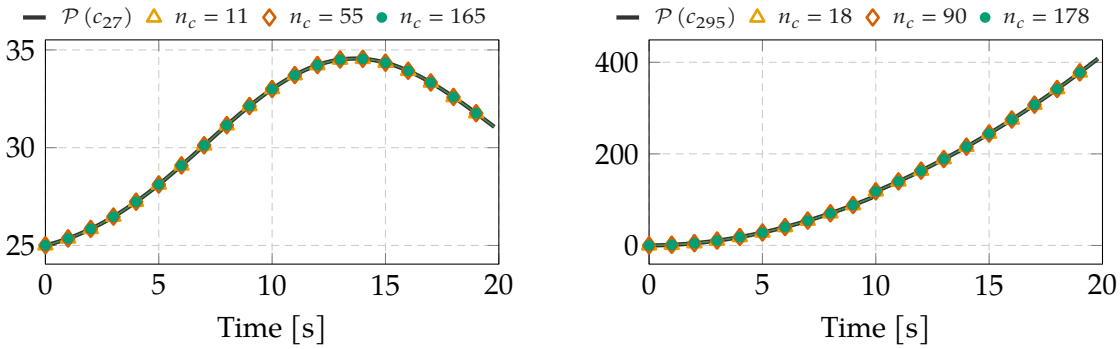


Figure 5.1: The plots display results for LLVM (left) and lrzip (right) for configurations c_{27} (left) and c_{295} (right), which were known for all models. The black line — shows the actual performance behavior of configurations c_{27} and c_{295}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

without interaction influences and with interaction influences. The first variant considers measurements where only individual configuration options affect performance, while the second includes the influence of their interactions.

Performance Data without Interaction Influences

First, we examine models μ_{t1}, μ_{t2}, and μ_{t3} built without considering the influences of interactions of configuration options. For both systems (LLVM and lrzip), we evaluate the predictions for selected configurations, distinguishing between configurations that were included in the sample set for all models, excluded from the sample set of model μ_{t1}, excluded from both the sample set of model μ_{t1} and μ_{t2}, and excluded from the sample sets for all models.

Known Configurations for All Models. The two plots in Figure 5.1 show the performance behavior for configurations that were included in the sample set for constructing models for all three sample sizes. For LLVM, the performance increases until around $t = 13$, where it reaches a peak and then decreases. For lrzip, the performance rises with a small jump at around $t = 10$. In both systems, the predicted performance exactly matches the actual behavior across all models, with all predicted values at each time point aligning with the measured values.

Unknown Configurations for Model μ_{t1}. Figure 5.2 shows configurations for LLVM and lrzip that were excluded from the sample set for constructing model μ_{t1} but included for the models μ_{t2} and μ_{t3}. For both systems, the predictions for the models μ_{t2} and μ_{t3} exactly match the actual performance behavior. In contrast, predictions of μ_{t1} deviate from the actual measurements.

For LLVM, the predicted trend follows the general shape of the actual function but is shifted downward. The jump at $t = 10$ is captured, and the rise until $t = 10$ is similar, though the increase is less steep than in the actual measurement. After $t = 10$, the predicted function roughly follows the trend of increasing slightly and then decreasing, but with a smaller amplitude.

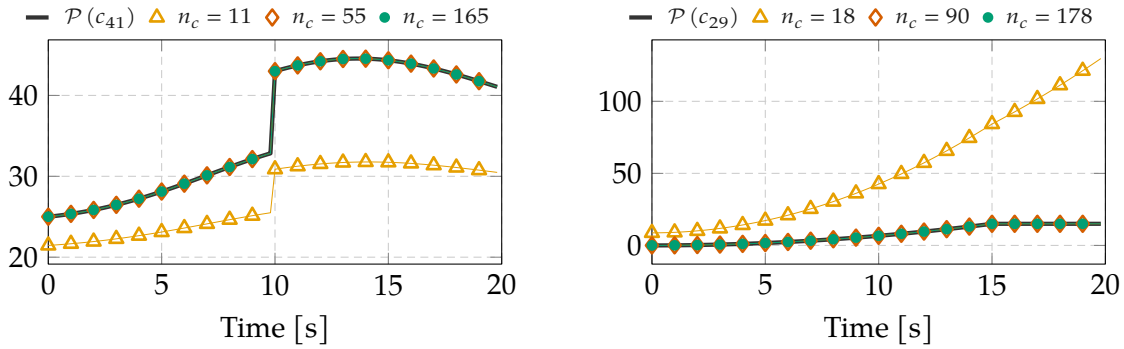


Figure 5.2: The plots display results for LLVM (left) and lrzip (right) for configurations c_{41} (left) and c_{29} (right), which were unknown for model μ_{t1}. The black line — shows the actual performance behavior of configurations c_{41} and c_{29}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

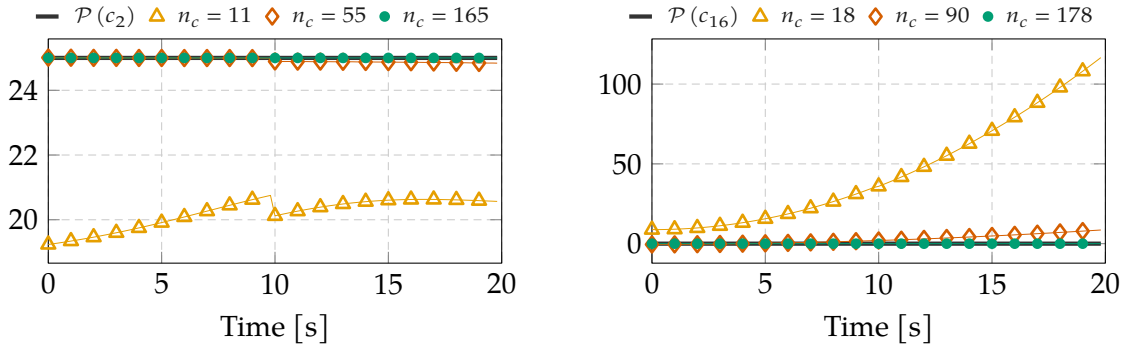


Figure 5.3: The plots display results for LLVM (left) and lrzip (right) for configurations c_2 (left) and c_{16} (right), which were unknown for models μ_{t1} and μ_{t2}. The black line — shows the actual performance behavior of configurations c_2 and c_{16}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

For lrzip, the prediction based on model μ_{t1} does not match the actual values and follows a different trend. The prediction is shifted upward and exhibits a steeper rise than the actual function. The actual performance increases slowly until around $t = 15$ and then remains constant. In contrast, the predicted function rises faster and continues to increase more gradually after $t = 15$. Despite these differences, the model captures the change at $t = 15$ and the general increasing trend up to this point.

Unknown Configurations for Models μ_{t1} and μ_{t2}. In Figure 5.3, we show two configurations, one for LLVM and lrzip, respectively, that were not included in the sample sets for constructing models μ_{t1} and μ_{t2}, but were part of the sample set for model μ_{t3}. For both systems, the predictions for model μ_{t3} match the actual performance values exactly. The predictions for the model μ_{t2} are also close to the actual values.

For LLVM, the predicted values for model μ_{t2} start at the correct value and remain nearly constant for a while, with a very slight decrease until $t = 10$. A small jump occurs at $t = 10$, followed by another period of roughly constant values with a minor decrease. While the jump should not be captured and the trend in the prediction is not exactly constant, the overall

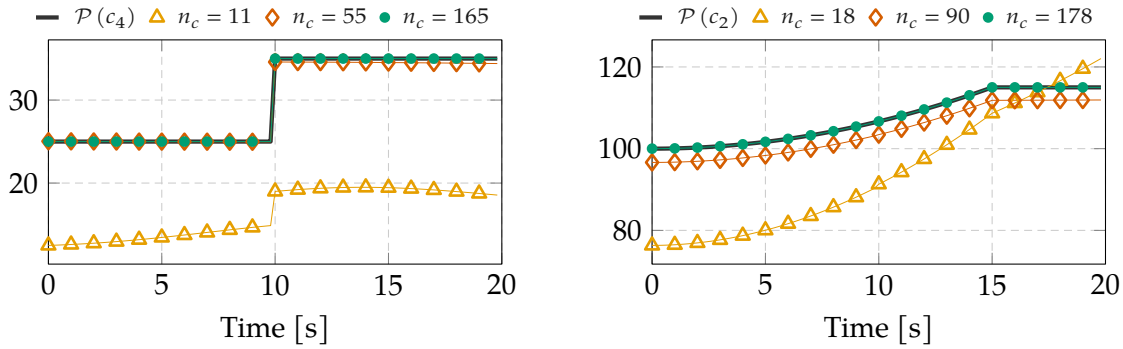


Figure 5.4: The plots display results for LLVM (left) and lrzip (right) for configurations c_4 (left) and c_2 (right), which were unknown for all models. The black line — shows the actual performance behavior of configurations c_4 and c_2, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

values are close to the actual measurements. For model μ_{t1}, the predicted trend deviates significantly with the values increasing until $t = 10$, having a downward drop, and then rising slightly in a bow-shaped pattern. Furthermore, the values are shifted downwards.

For lrzip, the predictions of model μ_{t2} initially align closely with the actual constant values near 0, but they gradually increase over time, so the trend is not constant as it is in the measured values. The predictions for μ_{t1} deviate strongly, showing a continuously increasing function that starts at a higher value and, therefore, differs in the trend.

Unknown Configurations for All Models. In Figure 5.4, we show two configurations, one for LLVM and lrzip, respectively, that were not included in the sample set for any model. For both systems, the predicted behavior of model μ_{t3} perfectly matches the actual values.

For LLVM, the predictions of model μ_{t2} are very close but not exactly correct. The constant value is captured until $t = 10$. The jump at $t = 10$ is captured but predicted slightly lower, and the values after the jump decrease slightly instead of remaining fully constant. For model μ_{t1}, the trend shows a clear deviation. The upward jump at $t = 10$ is captured, but before and after the jump, the values increase and then decrease rather than remaining constant. Additionally, the entire predicted sequence for μ_{t1} is shifted downward.

For lrzip we observe a similar result, where the predictions for μ_{t2} capture the overall trend accurately, showing the same increase until $t = 15$, followed by constant values. However, the only difference is that the sequence is shifted downward. The predictions for μ_{t1} show a more different trend, capturing the increase and the change at $t = 15$, but afterward the values continue increasing, but at a slower rate and linear, instead of remaining constant. Before $t = 15$, the increase is steeper, and the function is shifted downward, starting at around 40 instead of 100, creating a large gap. Additionally, a small jump appears at $t = 10$ in the predictions, which is not present in the actual measurement.

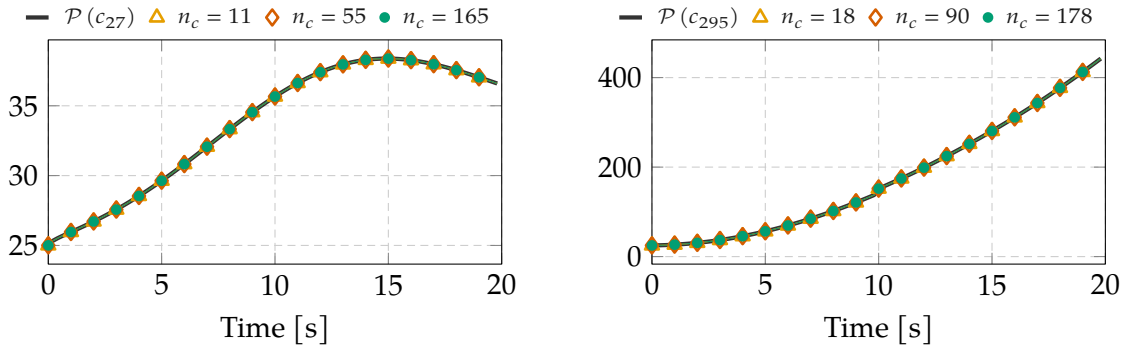


Figure 5.5: The plots display results for LLVM (left) and lrzip (right) for configurations c_{27} (left) and c_{295} (right), which were known for all models. The black line — shows the actual performance behavior of configurations c_{27} and c_{295}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

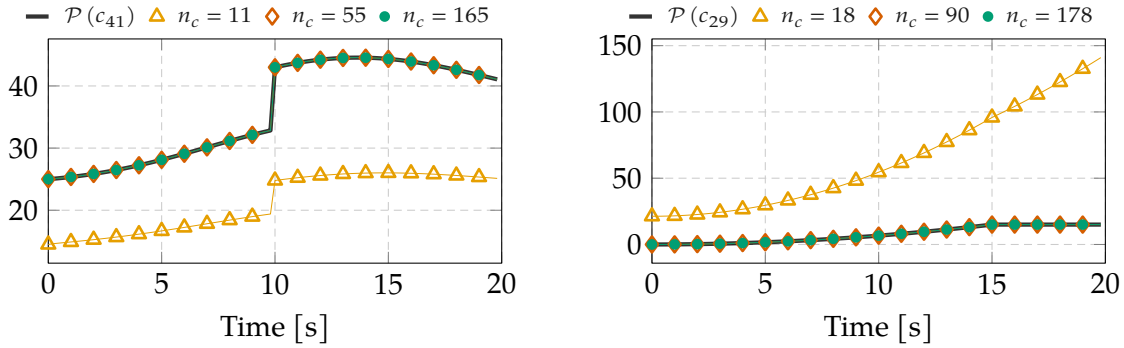


Figure 5.6: The plots display results for LLVM (left) and lrzip (right) for configurations c_{41} (left) and c_{29} (right), which were unknown for model μ_{t1}. The black line — shows the actual performance behavior of configurations c_{41} and c_{29}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

Performance Data including Interaction Influences

In this section, we examine the predictions of models built from performance measurements that incorporate the influences of interactions of configuration options. By analyzing the same configurations as before, we can directly compare how including influences for interactions affects the predicted performance behavior.

Known Configurations for All Models. Figure 5.5 shows the predicted performance values for two configurations, one for LLVM and lrzip, respectively, that were included in the sample sets for all models. For configuration 27 in LLVM, the actual performance sequence reaches higher values than in Figure 5.1, where influences of configuration option interactions were not considered in the seeded performance behavior. This indicates that configuration 27 includes an interaction of configuration options with an assigned influence, whereas for configuration 295 in lrzip, the performance function remains unchanged. For both systems, the predictions align perfectly with the actual performance values throughout the entire execution time and for all models.

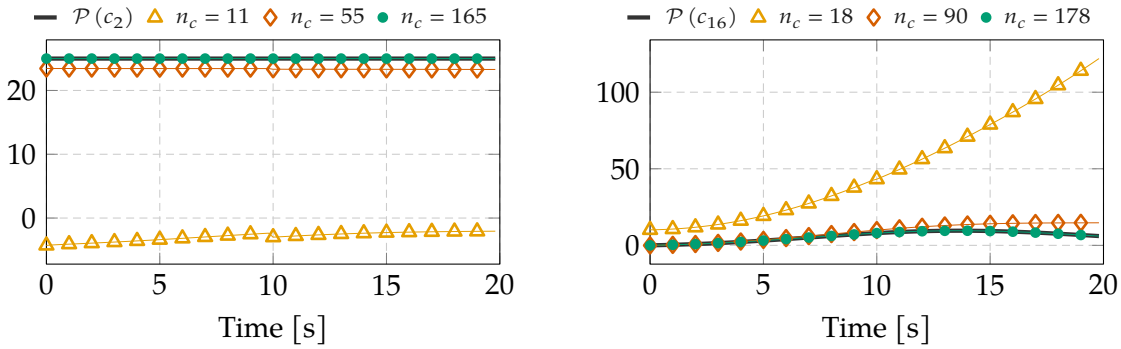


Figure 5.7: The plots display results for LLVM (left) and lrzip (right) for configurations c_2 (left) and c_{16} (right), which were unknown for models μ_{t1} and μ_{t2}. The black line — shows the actual performance behavior of configurations c_2 and c_{16}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

Unknown Configurations for Model μ_{t1}. Figure 5.6 shows two configurations that were included in the sample sets for constructing μ_{t2} and μ_{t3} but not in the sample set for μ_{t1}. For both systems, the predictions for the models μ_{t2} and μ_{t3} match the actual performance values perfectly.

For LLVM, the predictions for model μ_{t1} follow a similar trend but are shifted downward. The function still increases until $t = 10$, but with a less steep rate. At $t = 10$, the jump is captured, and the subsequent slight increase followed by a decrease is reproduced, though with smaller amplitude.

For lrzip, the predictions for μ_{t1} also deviate from the actual values. The trend is roughly similar, capturing the increase until $t = 15$ but with a much steeper rise. At $t = 15$, the predicted values reflect the change, but instead of remaining constant, they continue to increase at a slower rate. The entire sequence is additionally shifted upward.

For both configurations, the actual performance functions remain the same as in Figure 5.2, where influences of configuration option interactions were not included in the seeded performance behavior, indicating no effects from interactions. However, in both systems, the predictions of model μ_{t1} differ slightly from those in the case without interaction influences, following the same trend but appearing slightly shifted.

Unknown Configurations for Models μ_{t1} and μ_{t2}. Figure 5.7 shows configurations that were not included in sample sets for μ_{t1} and μ_{t2} but were part of the sample set for μ_{t3}. For both systems, the predictions for μ_{t3} perfectly match the actual performance values.

For LLVM, the predicted performance sequence of μ_{t2} is shifted downward and a small jump is visible at time point $t = 10$, but otherwise the constant trend is captured correctly. The sequence for μ_{t1} deviates more, being shifted much further downward and failing to capture the constant trend and instead increasing slowly. Additionally, there is an unintended downward jump at $t = 10$.

For lrzip, the predictions for μ_{t2} also capture the overall trend, but the increase occurs at a slightly different rate. The sequence starts at the correct value, but continues increasing for longer. The actual sequence rises until approximately $t = 13$ and then gradually decreases,

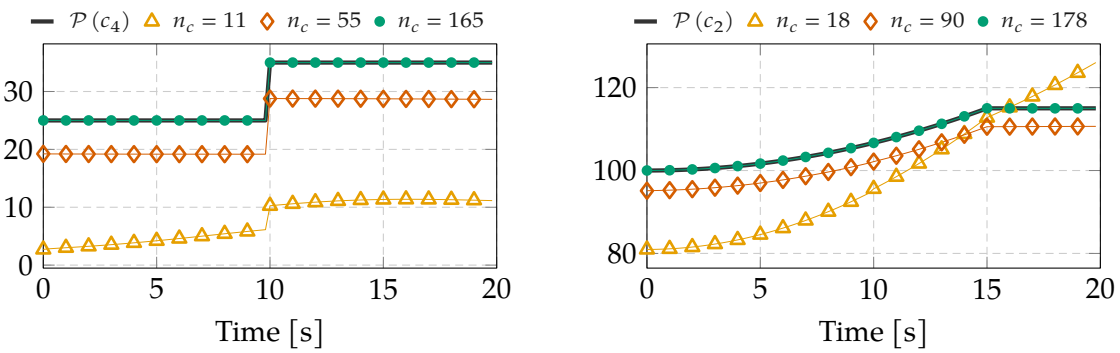


Figure 5.8: The plots display results for LLVM (left) and lrzip (right) for configurations c_4 (left) and c_2 (right), which were unknown for all models. The black line — shows the actual performance behavior of configurations c_4 and c_2, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3}.

whereas the predicted values keep increasing and start slowing around $t = 17$. For μ_{t1}, the values and trend deviate from the actual sequence, with the predicted sequence increasing steadily at a higher rate throughout the entire execution and being shifted upward.

For both configurations, the actual performance functions are identical to those without influences of interactions (c.f. Figure 5.3). For LLVM, the predictions for μ_{t2} are slightly shifted downward for seeded behavior with interaction degree 2. The difference is more pronounced for model μ_{t1}, where the predicted sequence shifts significantly downward, reaching negative values around -5 instead of 10. For lrzip, this effect does not occur, as the predictions remain consistent with those without interaction influences.

Unknown Configurations for All Models. For the configurations that are excluded from all sample sizes, Figure 5.8 shows that for both systems, the predictions for model μ_{t3} exactly match the actual sequence. In LLVM, the prediction for μ_{t2} capture the trend accurately, with the only difference being a downward shift. The sequence remains constant until $t = 10$, jumps up, and then stays constant. For model μ_{t1}, the predictions deviate but also the trend deviates more. The values increase until $t = 10$ instead of remaining constant as in the actual sequence, the jump is captured, and afterwards the values vary slightly rather than staying constant. The entire sequence is shifted downward.

For lrzip, the prediction for μ_{t2} also captures the trend correctly, but the sequence is shifted downward. For μ_{t1}, the predictions deviate, starting at lower values and increasing more steeply until $t = 15$, and after the change at $t = 15$ continue increasing linearly rather than remaining constant. Furthermore, a small upward jump appears at $t = 10$, which is not present in the actual measurements.

For both systems, the configurations exhibit the same actual performance sequences as in Figure 5.4 for seeded behavior without interaction influences. In LLVM, the predictions for μ_{t2} are slightly shifted downward compared to the case without interaction influences, though the overall trend is still captured. The predictions for μ_{t1} are shifted further downward, moving even farther from the actual measurements. Also for lrzip, μ_{t1} predicts a slightly upward shifted sequence when interaction influences are included, compared to the prediction without interaction influences.

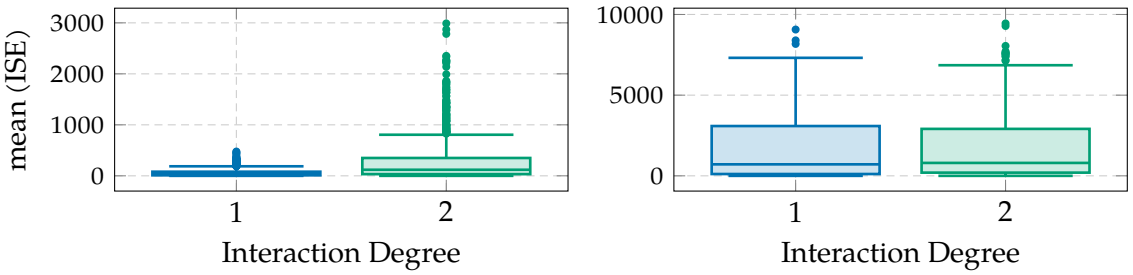


Figure 5.9: The left plot corresponds to LLVM and the right to lrzip. The boxplots show the distribution of the mean Integral Squared Error across all configurations for models μ_{t1}, without added noise. Within each plot, the left (blue) and right (green) boxplots illustrate the error distribution for interaction degrees 1 and 2.

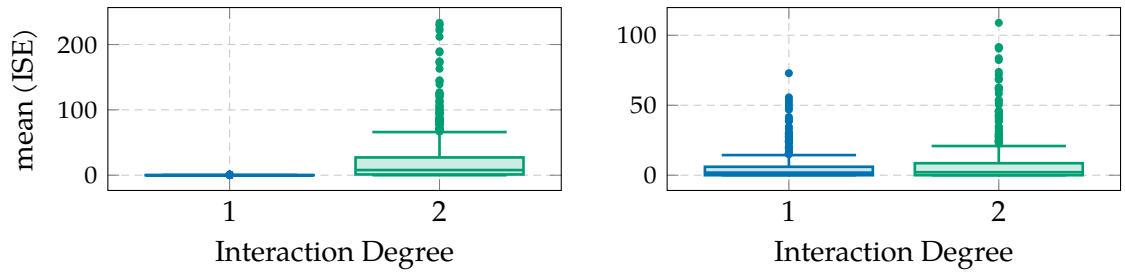


Figure 5.10: The left plot corresponds to LLVM and the right to lrzip. The boxplots show the distribution of the mean Integral Squared Error across all configurations for models μ_{t2}, without added noise. Within each plot, the left (blue) and right (green) boxplots illustrate the error distribution for interaction degrees 1 and 2.

Error Distribution

In the previous analysis, we focused on the predicted performance sequences of individual, representative configurations. In the following, we show aggregated results for all 1024 and 432 valid configurations for LLVM and lrzip, respectively. We quantify prediction errors using the mean Integral Squared Error (mean (ISE)). This metric summarizes the mean squared deviations between predicted and actual performance values over time into a single scalar value per configuration.

We visualize the distributions of these errors with boxplots for each sample size and further distinguish between models built with and without interaction influences. In each Figure, the left plot corresponds to LLVM and the right plot corresponds to lrzip, both separating seeded behavior with interaction degrees 1 (blue) and 2 (green).

Error Distribution for Model μ_{t1}. Figure 5.9 shows the distribution of the mean (ISE) for all configurations in both systems, based on models μ_{t1}. Each plot shows errors for models constructed with seeded behavior with and without interaction influences.

For LLVM, the errors for seeded behavior without interaction influences (Int Deg 1) are distributed between 0 and 80 (75% of the values), and 50% of the values are below 30 (median). The whiskers extend up to roughly 200, and there are some outliers reaching about 500. For seeded behavior with interaction influences (Int Deg 2), the distribution is wider. The median

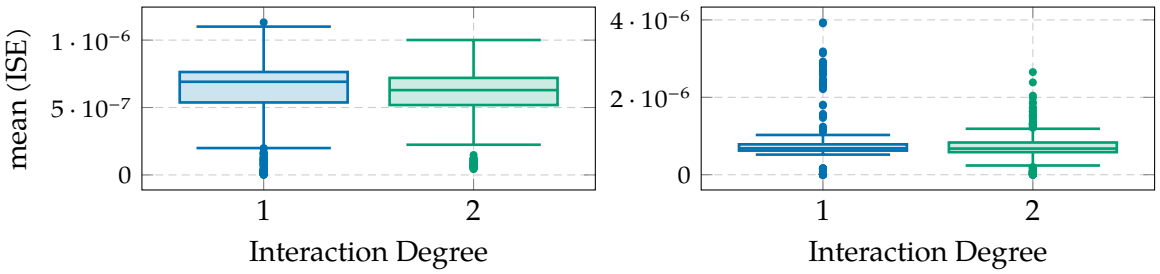


Figure 5.11: The left plot corresponds to LLVM and the right to lrzip. The boxplots show the distribution of the mean Integral Squared Error across all configurations for models μ_{t3}, without added noise. Within each plot, the left (blue) and right (green) boxplots illustrate the error distribution for interaction degrees 1 and 2.

is around 120, the interquartile range extends up to roughly 350, while the whiskers reach about 800. A small number of outliers appear, with errors as high as 3000.

For lrzip, the distribution is considerably broader. For seeded behavior without interaction influences, 75% of the errors lie between 0 and 3000. The median is at around 700, so half of the errors are smaller than 700. The upper whisker reaches to about 7500 and there are some outliers below 9000. For seeded behavior with interaction influences, the overall distribution is similar but slightly shifted downward, with outliers extending slightly higher.

Error Distribution for Model μ_{t2}. Figure 5.10 shows the distribution of the mean Integral Squared Errors for models μ_{t2} in both systems.

For LLVM and seeded behavior without interaction influences, the errors are concentrated very close to 0, with virtually no noticeable spread. When interaction influences are included, the errors become slightly larger with the interquartile range spanning from 0 to about 30, with a low median at around 10. The upper whisker extends to roughly 70, and a few outliers reach up to 250.

For lrzip, the errors for seeded behavior without interaction influences remain below 10 (75% of the values), with almost all errors staying under 15 except some outliers reaching up to 75. For seeded behavior with interaction influences, the distribution shifts upward slightly. Half of the errors lie below 3 (median) and all errors except outliers are below 25. Outliers appear at higher values, with errors reaching up to about 100.

Error Distribution for Model μ_{t3}. Figure 5.11 shows the distribution of the mean Integral Squared Error for both systems based on models μ_{t3}. For both systems and seeded behavior for both interaction degrees, the median error, the interquartile range, and even the whiskers remain extremely small, on the order of 10^{-6}, which can be considered negligible and effectively equal to 0.

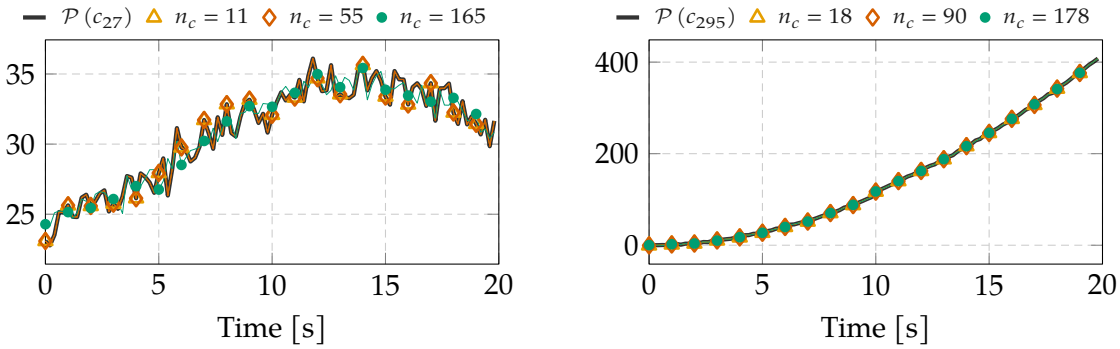


Figure 5.12: The plots display results for LLVM (left) and lrzip (right) for configurations c_{27} (left) and c_{295} (right), which were known for all models. The black line — shows the actual performance behavior of configurations c_{27} and c_{295}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

5.1.2 RQ2: Synthetic Noise Effect

In order to answer the second research question, we investigate the impact of synthetic noise by adding Gaussian noise with mean zero and standard deviation one to the performance measurements of each configuration. Each measurement point is perturbed by a random value drawn from this distribution. To isolate the effect of noise, we analyze the same configurations and scenarios as for the first research question.

Performance Data without Interaction Influences

We first built the models using performance data where only individual configuration options influence performance and examine the predictions for the same configurations as in the first research question, i.e., depending on whether they were included in the sample sets for models μ_{t1}, μ_{t2}, and μ_{t3}. The only difference compared to RQ1 with seeded behavior without interaction influences is the addition of synthetic noise.

Known Configurations for All Models. Figure 5.12 shows the performance behavior for two configurations included in the sample set for all models, with noise added to the measurement sequence. As in the noiseless case illustrated in Figure 5.1, the predicted performance values align with the actual measurements across all models. For both LLVM and lrzip, the predictions for the models μ_{t1} and μ_{t2} exactly hit the noisy measurements. In lrzip, where the noise is less pronounced, the predictions for model μ_{t3} also align with the noisy values. For LLVM, the predictions for μ_{t3} deviate more from the noisy measurements, as instead of hitting each noisy point exactly, they follow the overall trend and pass near the center of the noise, closer to the original noiseless values.

Unknown Configurations for Model μ_{t1}. For configurations not included in the sample set for models μ_{t1} but included for models μ_{t2} and μ_{t3}, Figure 5.13 shows the results for both systems. The predictions of model μ_{t1} still deviate from the actual measurements, similar to the case without noise in Figure 5.2, with values being shifted and following a slightly

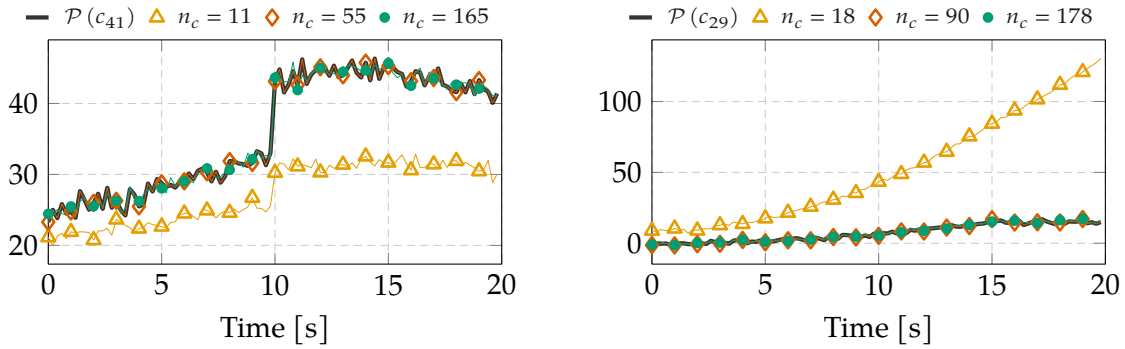


Figure 5.13: The plots display results for LLVM (left) and lrzip (right) for configurations c_{41} (left) and c_{29} (right), which were unknown for model μ_{t1}. The black line — shows the actual performance behavior of configurations c_{41} and c_{29}, while the colored markers (△, ◇, ⊛) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

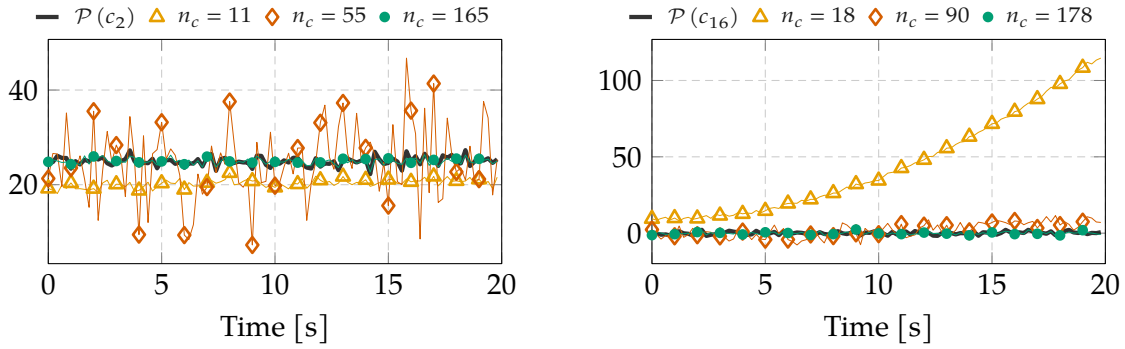


Figure 5.14: The plots display results for LLVM (left) and lrzip (right) for configurations c_2 (left) and c_{16} (right), which were unknown for models μ_{t1} and μ_{t2}. The black line — shows the actual performance behavior of configurations c_2 and c_{16}, while the colored markers (△, ◇, ⊛) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

different trend. However, the results also show that the models learn the noise, leading to noisy predictions. For μ_{t2}, the predictions exactly match the noisy measurements in both systems. For μ_{t3}, the predictions do not align with each noisy point but instead follow the overall trend and stay closer to the center of the noise, near the original noiseless values.

Unknown Configurations for Models μ_{t1} and μ_{t2}. Figure 5.14 shows configurations that were not included in sample sets for μ_{t1} and μ_{t2} but were part of the sample set for μ_{t3}. The predictions for model μ_{t3} do not exactly hit the noisy measurements but instead follow the center of the noise, close to the constant values for both LLVM and lrzip.

For model μ_{t2} in LLVM, the near constant behavior observed without noise (c.f. Figure 5.3) is now dominated by noise. The values fluctuate around the constant within a range of about ± 20, while still reflecting the underlying constant trend of the sequence. In lrzip, the predictions for μ_{t2} show less noise, oscillating around the constant sequence with smaller

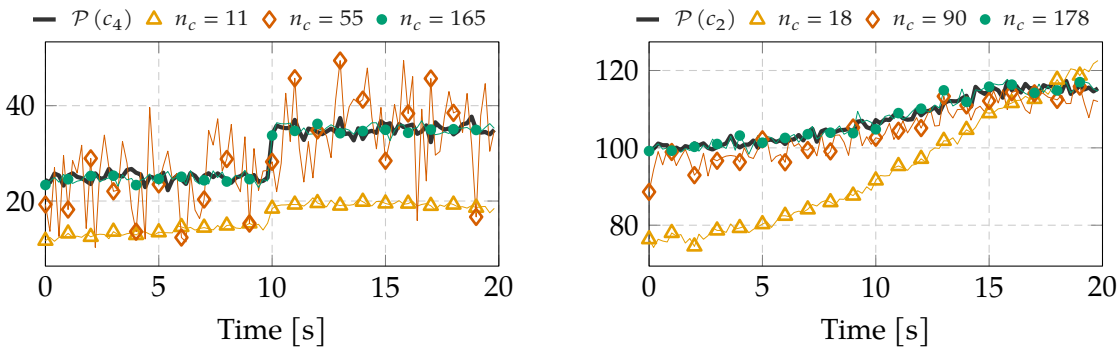


Figure 5.15: The plots display results for LLVM (left) and lrzip (right) for configurations c_4 (left) and c_2 (right), which were unknown for all models. The black line — shows the actual performance behavior of configurations c_4 and c_2, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

deviations. However, as in the noiseless case in Figure 5.3, the prediction still exhibits a slow upward trend.

For the predicted sequence of μ_{t1} in LLVM, the trend of the sequence has changed. Instead of the incorrect increasing function with a jump illustrated in Figure 5.3, the predictions form a nearly constant sequence, slightly shifted downward and added with noise. In lrzip, the predictions for μ_{t1} remain an increasing function rather than constant, but now also include some noise.

Unknown Configurations for All Models. Figure 5.15 shows the predictions for two configurations that are not included in the sample set for any model. The predictions for μ_{t3} are slightly noisy, but do not exactly match the noisy measurements, remaining very close to the original noiseless sequences.

In LLVM, the predictions for μ_{t2} are strongly dominated by noise, yet the overall trend is still visible, as the constant behavior before and after the jump at $t = 10$ can be recognized despite heavy fluctuations. For lrzip, the predictions for μ_{t2} follow the same shifted trend as in the noiseless case in Figure 5.4, but with added noise.

For model μ_{t1}, both systems reproduce the same behavior as in the noiseless case of Figure 5.4, now with added noise. In LLVM, the increasing trend with the jump remains shifted downwards, though slightly perturbed by fluctuations, and in lrzip the same increasing sequence persists, again with added noise.

Performance Data including Interaction Influences

To evaluate the impact of interaction influences on noisy measurements, we built models from performance data that include the influences of both individual configuration options and their interactions. The interactions influencing performance are the same as those considered in the first research question. We analyze the predictions for the same configurations as in RQ1 with performance data including interaction influences, allowing a comparison of how added noise affects the predicted performance behavior.

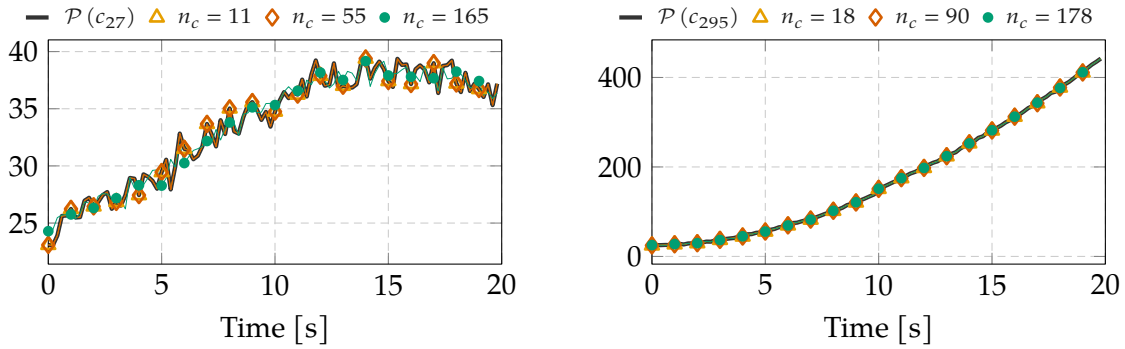


Figure 5.16: The plots display results for LLVM (left) and lrzip (right) for configurations c_{27} (left) and c_{295} (right), which were known for all models. The black line — shows the actual performance behavior of configurations c_{27} and c_{295}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

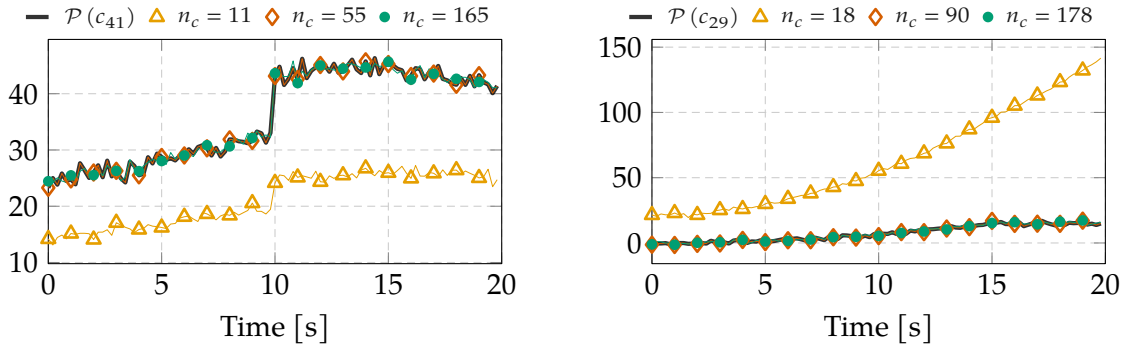


Figure 5.17: The plots display results for LLVM (left) and lrzip (right) for configurations c_{41} (left) and c_{29} (right), which were unknown for model μ_{t1}. The black line — shows the actual performance behavior of configurations c_{41} and c_{29}, while the colored markers (△, ◇, ●) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

Known Configurations for All Models. In Figure 5.16, for configurations included in the sample set for all models, the predictions for models μ_{t1} and μ_{t2} exactly match the noisy measurements for both systems. In lrzip, where the noise is less pronounced, the predictions for model μ_{t3} also align with the exact values. For LLVM, the predictions for model μ_{t3} deviate from the noisy measurements and are closer to the original noiseless values demonstrated in Figure 5.5, following the overall trend rather than hitting each noisy point exactly. Deviations due to noise are still present but less extreme than in the actual noisy measurements, capturing the general trend without reproducing every fluctuation.

Unknown Configurations for Model μ_{t1}. In Figure 5.17, the predictions for configurations excluded from the sample set for constructing μ_{t1} closely resemble those in the noiseless case illustrated in Figure 5.6. For both systems, the predicted sequences remain the same but with added noise, which is more pronounced in LLVM. The predicted sequence for μ_{t1} is shifted

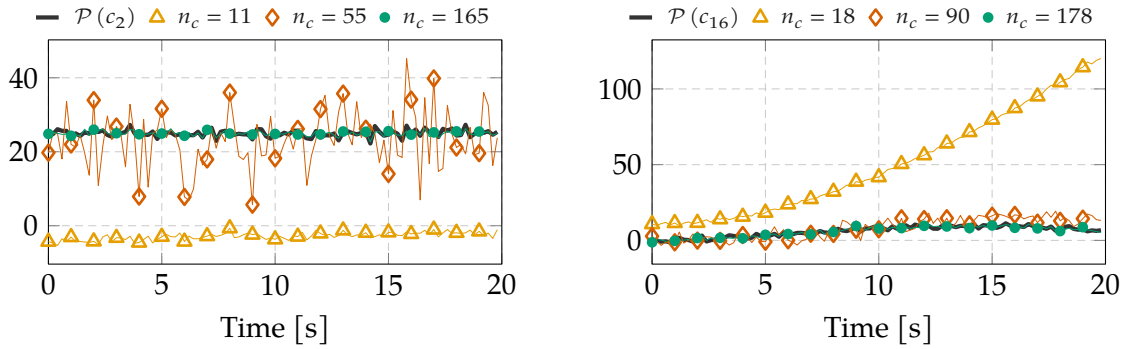


Figure 5.18: The plots display results for LLVM (left) and lrzip (right) for configurations c_2 (left) and c_{16} (right), which were unknown for models μ_{t1} and μ_{t2}. The black line — shows the actual performance behavior of configurations c_2 and c_{16}, while the colored markers (△, ◇, ⊛) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

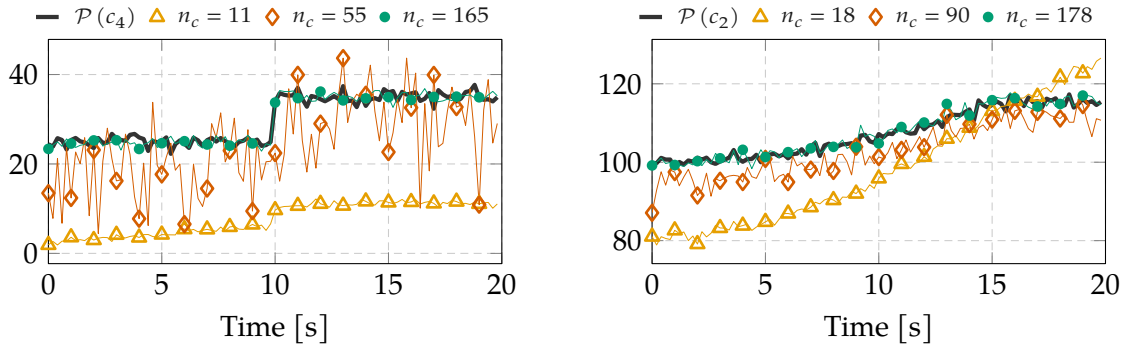


Figure 5.19: The plots display results for LLVM (left) and lrzip (right) for configurations c_4 (left) and c_2 (right), which were unknown for all models. The black line — shows the actual performance behavior of configurations c_4 and c_2, while the colored markers (△, ◇, ⊛) represent the predicted performance from models μ_{t1}, μ_{t2}, and μ_{t3} constructed with added noise.

downward, for μ_{t2} the predictions match the noisy measurements exactly, and predictions for μ_{t3} deviate slightly from the exact measurements. In LLVM, the predictions for μ_{t3} are closer to the underlying noiseless sequence, lying near the center of the noise.

Unknown Configurations for Models μ_{t1} and μ_{t2}. Figure 5.18 shows the predicted sequences for two configurations that are excluded in the sample set for models μ_{t1} and μ_{t2} and included for model μ_{t3}. The predictions for model μ_{t1} remain the same as in the noiseless case in Figure 5.7 but with added noise. In LLVM, the sequence still exhibits the same downward shift and increasing trend before and after the jump at $t = 15$. Similarly, in lrzip the predictions for μ_{t1} continue to show an increasing trend, now with noise added. For model μ_{t2} in LLVM, predictions are dominated by large noise fluctuations, though the overall constant trend remains visible. In contrast, predictions of μ_{t2} in lrzip follow the same trend as in the noiseless case (c.f. Figure 5.7), with comparatively smaller noise deviations. For model μ_{t3}, predictions in both systems closely match the actual values, deviating from the exact noisy points slightly.

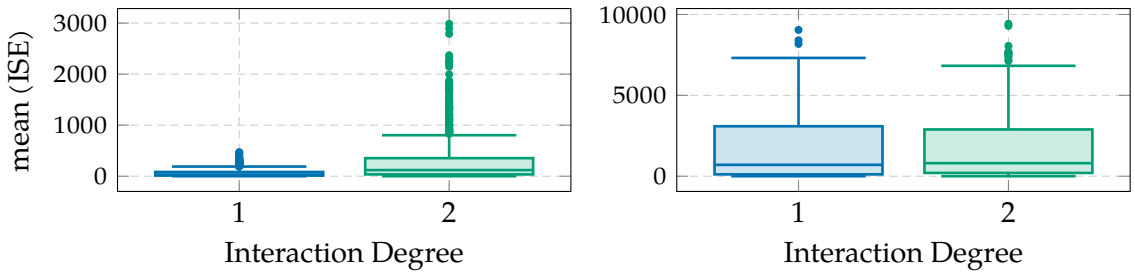


Figure 5.20: The left plot corresponds to LLVM and the right to lrzip. The boxplots show the distribution of the mean Integral Squared Error across all configurations for models μ_{t1}, with added noise. Within each plot, the left (blue) and right (green) boxplots illustrate the error distribution for interaction degrees 1 and 2.

Unknown Configurations for All Models. When the configurations are unknown for all models, the predictions remain largely similar to the noiseless case shown in Figure 5.8 but with added deviations due to noise, as shown in Figure 5.19. For both systems, the predictions for μ_{t1} follow the same sequences as in the noiseless case, now overlaid with some noise. In LLVM, predictions for μ_{t2} are strongly affected by noise, but the overall trend remains visible, with the constant behavior before and after the jump at $t = 10$ still recognizable despite large fluctuations. In lrzip, the predictions for μ_{t2} also follow the same shifted trend as in the noiseless case (c.f. Figure 5.8) but with added noise, though the amount of noise is much smaller than in LLVM. For model μ_{t3}, predictions in both systems closely follow the actual values, showing only minor noise without exactly matching the noisy measurements.

Error Distribution

To analyze the effect of synthetic noise on prediction accuracy across all configurations, we summarize the prediction error between the predicted and actual performance sequence using the mean Integral Squared Error. The distribution of the errors is visualized with boxplots for each system and sample size, distinguishing between models built with seeded behavior with and without interaction influences. For each plot, LLVM is shown on the left and lrzip is shown on the right, for both interaction degrees, and for the corresponding sample size.

Error Distribution for Model μ_{t1}. Figure 5.20 shows the mean Integral Squared Error across all configurations for models μ_{t1} constructed with noisy measurements. For LLVM, considering seeded behavior with individual options, 75% of the errors lie between 0 and 80, with whiskers reaching about 200 and some outliers up to 500. For seeded behavior with interaction influences, the median is at around 120, the interquartile range reaches up to roughly 350 and whiskers extend to approximately 800. Outliers appear, with maximum errors reaching around 3000. For lrzip and seeded behavior without interaction influences, 75% of the errors lie between 0 and 3000, and the median is at around 700, so half of the errors are below 700. The upper whisker reaches roughly 7500 and there are some outliers reaching up to 9000. For seeded behavior with interaction influences, the overall distribution remains similar, but additional outliers appear and extend slightly higher.

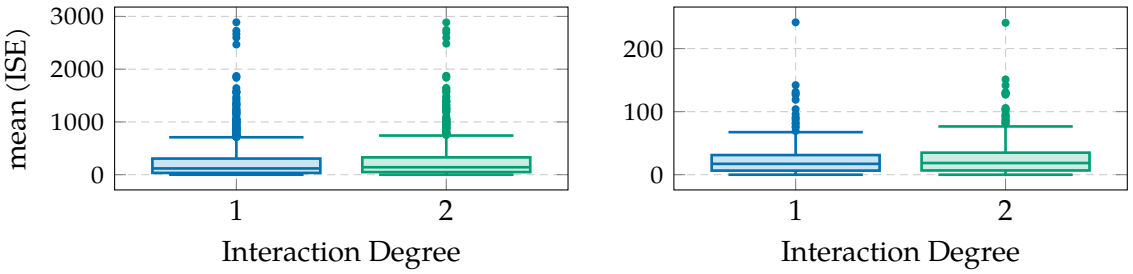


Figure 5.21: The left plot corresponds to LLVM and the right to lrzip. The boxplots show the distribution of the mean Integral Squared Error across all configurations for models μ_{t2}, with added noise. Within each plot, the left (blue) and right (green) boxplots illustrate the error distribution for interaction degrees 1 and 2.

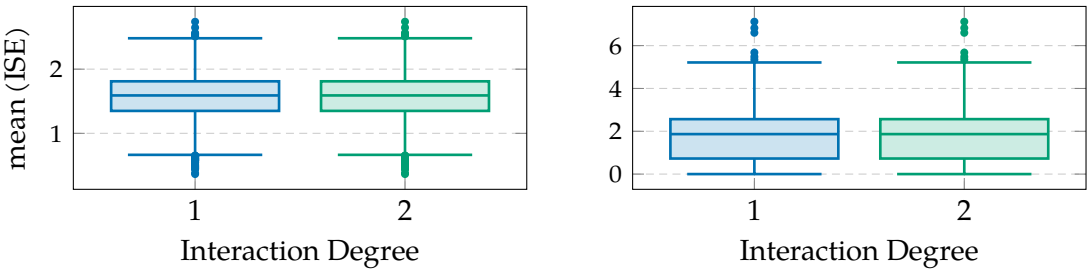


Figure 5.22: The left plot corresponds to LLVM and the right to lrzip. The boxplots show the distribution of the mean Integral Squared Error across all configurations for models μ_{t3}, with added noise. Within each plot, the left (blue) and right (green) boxplots illustrate the error distribution for interaction degrees 1 and 2.

Error Distribution for Model μ_{t2}. Figure 5.21 shows the distribution of the prediction error for the models μ_{t2}. For LLVM and seeded behavior without interaction influences, approximately half of the prediction errors are below 120 (median). The interquartile range extends up to 300, while the whiskers reach about 700. Outliers extend up to around 3000. For seeded behavior with interaction influences, the distribution is similar but slightly shifted upward. For lrzip and seeded behavior without interaction influences, the errors are distributed below roughly 30 (75% of the values). The upper 25% of errors reach about 70. Outliers are present, most extending up to 150, with a few reaching approximately 250. For seeded behavior with interaction influences, the distribution is again similar but slightly shifted upward.

Error Distribution for Model μ_{t3}. The distribution of the prediction error for models μ_{t3} is shown in Figure 5.22. For LLVM and seeded behavior without interaction influences, all errors range from 0 to 2.5, with 75% of the errors lying below 2. Outliers appear below 0.8 and above 2.5 up to around 3. When interaction influences are included in the seeded behavior, the overall distribution remains the same. For seeded behavior without interaction influences in lrzip, the errors lie below 2.5 (75% of the values), with the whiskers extending up to 5.5 and outliers extending to nearly 8. Including interaction influences in the seeded behavior results in a similar distribution.

5.2 Discussion

In this section, we discuss the results of our seeded experiments, focusing on how different factors in model building affect prediction accuracy. We examine the relationship between the sample size used for model construction and the accuracy of the models' predictions. We also discuss how including influences to interactions between configuration options in the seeded performance data influences accuracy. Finally, we assess the impact of adding synthetic noise on prediction accuracy compared to models built from measurements without noise.

5.2.1 RQ1: Prediction Accuracy

In this section, we discuss implications and possible learnings from the results for the first research question. We analyze factors that affect prediction accuracy, identify patterns across different sample sizes, and examine how the inclusion of interaction influences impacts the predictions.

Performance Data without Interaction Influences. As shown in Figure 5.1 and Figure 5.5, for configurations included in the sample sets for all models μ_{t1}, μ_{t2}, and μ_{t3}, the predictions across all models perfectly match the actual performance values. This suggests that, when a configuration is part of the training data, the model can reliably learn and reproduce its performance behavior.

In Figure 5.2, we observe that predictions for models μ_{t2} and μ_{t3} remain accurate, since the configuration was still present in their sample sets. In contrast, the predictions for model μ_{t1} deviate from the actual sequences in both systems. For LLVM, this deviation is largely a downward shift with only minor differences in trend, suggesting that some influences of configuration options were learned correctly but others were missing due to the small sample size. For lrzip, both the values and the trend deviate more strongly, indicating that the sample set used to build model μ_{t1} did not cover enough configuration options.

For configurations only included in model construction of μ_{t3} as in Figure 5.3, predictions of μ_{t3} remain perfect. Predictions from models μ_{t2} are close to the actual values, with only slight deviations, while those from model μ_{t1} fail to capture the correct trend. This shows that sample size t2 already obtains enough information to assign the correct influences, while sample size t1 fails to see enough configurations to learn the influences accurately.

Finally, for configurations excluded from the sample sets of all models as in Figure 5.4, the predictions for μ_{t3} still align perfectly with the measured values. This suggests that, even without having a configuration in the sample set, sample size t3 provides sufficient coverage to infer the correct influences and predict unseen configurations accurately. The predictions for μ_{t2} again capture the correct trend with only minor shifts, while predictions for μ_{t1} remain less accurate, often capturing the general shape but at incorrect values.

Performance Data including Interaction Influences. Seeded behavior with influences for configuration option interactions do not affect predictions when a configuration is included in the sample set, as the sequences are captured perfectly. For unseen configurations, models

μ_{t3} still predict the performance sequence accurately, indicating that for model construction sufficient configurations were included to capture the additional influences. Predictions from models μ_{t1} and μ_{t2} are slightly worse for seeded behavior with interaction degree 2 influencing the performance behavior. Even when assigning influences to interactions does not change the actual performance function of a configuration, it can still affect the predictions of the models. The primary effect is a shift in the predicted sequences, while the overall trends remain accurate. This effect is particularly noticeable in LLVM, where the shifts move predictions further from the actual values, resulting in larger prediction errors. In contrast, the shifts are present, but much less pronounced in lrzip, indicating that interaction influences have a minor impact. This suggests that the inclusion of interaction influences can affect the assignment of individual configuration options during model construction, leading to different predictions even when the interactions do not directly influence the performance of a specific configuration.

Error Distribution. To summarize the prediction accuracy across all configurations, we analyze the mean Integral Squared Error distributions for the different sample sizes, c.f., Figure 5.9, Figure 5.10, and Figure 5.11. The boxplots reveal that, as the sample size increases, the prediction errors decrease for both systems. Additionally, seeded behavior with interaction degree 2 affects the accuracy as errors are generally higher when interaction influences are included.

For LLVM, Figure 5.9 shows that all prediction errors for model μ_{t1} and seeded behavior with interaction degree 1 are in a range below 200. These errors may arise because the sample set for constructing μ_{t1} is too small to cover all individual configuration options, preventing the model from fully capturing their effects. The same issue applies when considering seeded behavior with interaction degree 2, where higher errors ranges occur, reaching around 800 and having outliers reaching errors of up to 3000. In this case, not only some individual configuration options might be missing in the sample, but many interactions might also be unobserved, preventing their influences from being learned. Consequently, for model μ_{t1}, prediction errors are higher with seeded behavior with interaction degree 2 than for interaction degree 1, which may be due to the limited coverage of the additional interaction influences.

A similar pattern can be observed for lrzip. Figure 5.9 illustrates that for model μ_{t1}, the errors are higher in the range of thousands due to the small sample size, which leaves many influences unobserved. The errors are even larger than for LLVM, possibly because its configuration space is more complex due to constraints across configuration options. This complexity could also make the sampled configurations less representative, as random sampling fails to cover enough configurations with different options enabled. Similar to LLVM, the errors for seeded behavior with interaction degree 2 are higher than for interaction degree 1. While the overall distribution is similar, the main difference is that with interaction influences, outliers are reaching higher error values.

As the sample size increases, Figure 5.10 shows that prediction errors decrease for models μ_{t2} for seeded behavior with interaction degree 1 and 2. In LLVM, the errors for seeded behavior with interaction degree 1 are already close to 0, indicating that the sample size is sufficient to observe all individual configuration options and capture their effects. For seeded behavior with interaction degree 2, most errors remain below 70, although outliers can reach

more than 200. This shows that while the influences of individual options are captured, some interactions are still unobserved, which leads to higher errors.

The same trend is observed for lrzip. Errors for seeded behavior with interaction degree 1 are no longer 0 but remain small (< 15), reflecting improved predictions due to better coverage. For seeded behavior with interaction degree 2, the error distribution is slightly higher, with additional outliers reaching larger values. Overall, predictions from model μ_{t2} achieve reduced errors, though they still do not fully capture the impact of interaction influences.

For models μ_{t3}, Figure 5.11 shows that the errors for both systems, with seeded performance behavior 1 and 2, are extremely close to 0 and can be considered negligible. This indicates that the sample size is sufficient to capture all relevant influences, including the influence of configuration option interactions, enabling accurate predictions even for unseen configurations.

The boxplots reinforce the observations from the example sequences over execution time. As the sample size increases, the corresponding models are able to learn the correct influences more accurately, resulting in more precise performance predictions. Assigning influences to configuration option interactions negatively affects prediction accuracy, particularly for smaller sample sizes, as model construction does not include enough configurations to learn the influences of interactions. Additionally, for lrzip, the differences in prediction errors between models built with seeded behavior with interaction degree 1 and 2 are smaller than for LLVM. In lrzip, the overall error distribution remains similar, with only outliers showing larger errors, whereas for LLVM, the entire distribution shifts upward when interaction influences are included.

RQ1 (Prediction Accuracy): The accuracy of the predicted sequences improves with increasing sample size, as larger samples capture more relevant influences and allow accurate predictions even for unseen configurations. With smaller sample sizes, the overall performance trends remain largely preserved, but some influences are missed, which is further amplified when interaction influences are included.

5.2.2 RQ2: Synthetic Noise Effect

For the second research question, we investigate how adding synthetic noise to our measurements affects the prediction accuracy of our models. In particular, we discuss how noise impacts the models ability to capture the underlying noiseless performance behavior.

Performance Data without Interaction Influences. When a configuration is included in the sample set for μ_{t1} and μ_{t2}, the predictions match the noisy measurements exactly. If a configuration is not included in the sample set for model μ_{t3}, e. g., in Figure 5.13, Figure 5.14, and Figure 5.15, the predictions resemble those in the noiseless case, showing the same incorrect shifts and trends. However, the added synthetic noise introduces additional deviations in the predictions for both systems.

The predictions of μ_{t2} align perfectly with the noisy measurements when the configuration is part of the sample set as shown in Figure 5.12 and Figure 5.13. When the configuration is

not included (c.f. Figure 5.14 and Figure 5.15), the predictions still capture the general trend but are heavily influenced by noise, especially in LLVM, where deviations are larger. This behavior suggests overfitting, as the model attempts to reproduce the noise as if it were part of the true performance trend, leading to predictions that are distorted by noise. Another possible explanation is that the sample set for μ_{t2} happened to include configurations with particularly strong noise. In that case, the model would capture these deviations, amplifying the effect. With a different sample set, the predictions might therefore appear less noisy.

In contrast, predictions of models μ_{t3} do not match every noisy point but instead follow the overall trend and remain closer to the center of the noise, near the original noiseless values. This indicates that with more included configurations, the model captures the underlying performance behavior more robustly, avoiding overfitting to noise. In particular, it learns that the noise oscillates around a mean of 0 and therefore stays centered around the noiseless sequence. Thus, larger sample sizes reduce the effect of overfitting on noise and make it possible to regularize the predictions towards the true underlying performance function.

Performance Data including Interaction Influences. When noise is added to the measurements, seeded behavior with interaction degree 2 shows the same patterns as in the noiseless case. For models μ_{t3}, the predictions still closely follow the actual measurements, regardless of whether the configuration was included in the sample set. Predictions for models μ_{t1} and μ_{t2} resemble those for seeded behavior with interaction degree 1. As in the noiseless case, their predictions are slightly worse than for seeded behavior with interaction degree 1, as sequences tend to be more shifted from the actual performance values. Nevertheless, the models still capture the overall performance trends, even when configurations are not part of the sample set.

Error Distribution. We summarize the prediction accuracy of models trained on noisy measurements using the mean Integral Squared Error for each sample size, shown in Figure 5.20, Figure 5.21, and Figure 5.22. The error distributions for model μ_{t1} are essentially identical to those in the noiseless case as illustrated in Figure 5.9. This occurs because predictions for configurations included in the sample set match the noisy measurements exactly and thus no errors are introduced, as in the noiseless case. For configurations not present in the sample set, the model predicts the same, possibly incorrect, sequence as in the noiseless case, with noise simply superimposed. Consequently, the predicted sequences and the corresponding errors remain largely unchanged for both systems and for models constructed with seeded behavior for both interaction degrees.

For models μ_{t2}, adding synthetic noise substantially increases prediction errors compared to the noiseless case. This effect is observed in both LLVM and lrzip and occurs for seeded behavior with both interaction degrees. The main reason for these higher errors is likely overfitting to noise. The model interprets noisy measurements as part of the true performance behavior, leading to highly fluctuating predictions and consequently higher errors. This effect was also observed in the plots with the predicted sequence, where predictions for μ_{t2} appeared very noisy and deviated significantly from the actual performance. However, consistent with our previous findings, both systems show increased errors and additional outliers for seeded behavior with interaction degree 2 compared to degree 1.

For models μ_{t3}, the errors are now very small, staying below 3 for LLVM and below 6 for lrzip, as shown in Figure 5.22. These errors are higher than in the noiseless case as shown in Figure 5.11, where they were negligible. However, this is expected as the goal is not to fit the noise but to capture the underlying performance behavior. As seen in the plots with the predicted sequences over execution time, predictions of model μ_{t3} follow the overall trend and align closely with the center of the noise, i. e., near the original noiseless values. Consequently, some error remains because the predictions are not identical to the noisy measurements, which is an intended deviation. Overall, the errors remain very small, showing that the predictions closely reflect the true performance behavior. As in the noiseless case, sample size t3 is sufficient to capture all influences while regularizing the predictions toward the underlying performance function, thus avoiding overfitting to noise.

RQ2 (Synthetic Noise Effect): For smaller sample sizes the model tends to overfit to the noisy measurements, resulting in highly fluctuating predictions and increased errors. Larger samples capture the underlying performance trends, regularizing the noise and yielding accurate predictions similar to the noiseless case.

5.3 Threats to Validity

In this section, we discuss potential threats to validity of our experimental setup and results and the measures we took to mitigate them. We distinguish between internal validity, which concerns the correctness and reliability of our experimental setup and model evaluation, and external validity, which relates to the generalizability of our findings to other systems and noise conditions.

5.3.1 Threats to Internal Validity

To evaluate the accuracy of our models and establish the expected outcomes, we conduct experiments using ground-truth performance functions to simulate the behavior of configurations over time. We seed these performance functions for the configuration space of two real-world configurable software systems, LLVM and lrzip. This setup enables a systematic evaluation of the accuracy of our models under controlled conditions, allowing to specify which configuration options and interactions are assigned influences on the overall performance behavior.

In real-world systems, the number of configuration options influencing performance can vary. Some systems may have only a few options that significantly impact performance, while others may have many. The models' prediction accuracy may depend not only on the number of options influencing performance but also on the constraints on the options, e. g., whether options are mandatory or optional. This variability poses a threat to our results on prediction accuracy, since conclusions based on a specific assignment of configuration options may not hold for other assignments. To mitigate this threat and reflect the variability in our experiments, we vary the percentage of configuration options with assigned influences on

performance and evaluate how this affects the prediction accuracy of our models. Specifically, we assign influences to 25%, 50%, and 100% of options.

Another threat arises from the fact that, in practice, interactions between configuration options can significantly affect performance. The number of interactions and the extent to which they affect performance can vary across systems. To mitigate this threat, we consider each influence assignment with two variants, one including influences for interactions and one without. Including interaction influences increases the complexity of the relationships the model must capture to predict the performance of configurations accurately, which may impact its prediction accuracy.

Between the variants including interaction influences and those without, differences in which configuration options receive influences and the specific influences assigned to them could affect the comparison between the two variants. This threat can hinder the assessment of the isolated influence of interactions on the prediction accuracy of our models. To isolate the effect of interaction influences on the prediction accuracy, within a scenario and assignment, the variant without interaction influences is a subset of the variant with interaction influences. That means, we use the same configuration options and the same individual influences, adding only the interaction influences. Within each system, the same interactions of configuration options are assigned influences across all scenarios to ensure consistency.

When assigning functions to configuration options and interactions, a further threat arises because performance behavior can differ across systems and between different configuration options. That means, in real-world systems, there may be a range of different performance behaviors, and relying on only one assignment of functions would not capture this variety. To address this, each experiment is repeated with two independent assignments of configuration options and interactions to assign influences to. In these assignments, the corresponding functions representing the influences are also reassigned. Results for all scenarios are provided in the Appendix A.

In constructing our models, we formulate constraints from the measurements of each configuration in the sample set. Consequently, the number of configurations included in the sample set directly impacts prediction accuracy and relying on a single sample size could influence the results of our experiment. To address this, we evaluate how different sample sizes impact the models' ability to capture the underlying performance behavior. We vary the sample size for randomly selected configurations used in model construction, considering three sample sizes corresponding to the size of t-wise coverage sets with $t \in \{1, 2, 3\}$.

To investigate the isolated effect of synthetic noise, we repeat all experiments using the same configuration options and interactions with identical assigned influences, but add Gaussian noise to the performance function of each configuration. In the experiments with noisy measurements, we observe that for models μ_{t1} and μ_{t2} the predictions tend to overfit to noise and try to capture noise as part of the underlying performance behavior. The models μ_{t3} are able to regularize the noise and stay closer to the center of the noise distribution, and thus closer to the original noiseless performance function. However, the construction of model μ_{t3} may have already included an unrealistically large number of configurations, particularly in lrzip, which has a smaller configuration space, where 178 out of 432 configurations were already included and only a few configuration options had assigned influences. This suggests that future work could explore additional sample sizes and investigate methods to regularize noise for smaller sample sizes.

5.3.2 Threats to External Validity

We use the configuration spaces of two real-world subject systems, LLVM and lrzip, to assign influences to their configuration options and interactions. These systems differ in the number of configuration options and their constraints. Specifically, LLVM is minimally constrained, with only a root configuration option and all others optional, whereas lrzip has a more complex feature model with constraints between options. The models' performance might depend on characteristics of the subject system, such as the number of configurations, configuration options or the constraints between them. This poses a threat to the generalizability of our results, as conclusions drawn from one system might not hold for others. By selecting two systems with different characteristics, where LLVM has minimal constraints and lrzip has a more complex configuration space, we mitigate this threat and can observe how prediction accuracy varies across the configuration spaces.

Since performance behavior can vary widely across configurations and systems in real-world settings, we need to evaluate how well our model captures different types of performance behavior. A potential threat is that the evaluation may not cover the entire range of potential performance patterns that might occur in real-world systems, including both simple and complex behaviors. To mitigate this threat, we use a diverse set of functions including linear, constant, polynomial, stepwise, and periodic functions (c.f. Table 4.2), ensuring the evaluation captures a wide range of realistic performance behaviors.

In real-world systems, performance measurements can be affected by various sources of noise, which can distort the actual performance behavior and make it challenging to capture the underlying performance function. Consequently, another threat to external validity arises from the distribution of noise added to the performance functions, since real-world measurements noise can occur in different types and distributions. We chose Gaussian noise with a mean of 0 and a standard deviation of 1.0, which is relatively high compared to the actual performance values. We selected a standard deviation of 1.0 to simulate challenging conditions, reflecting the possibility of high noise in real-world scenarios. However, in practice, both the type and distribution of noise may differ. To explore how predictions change under less pronounced noise, we conducted a first experiment using a reduced standard deviation of 0.05 for model μ_{t2}, which had mostly shown distorted predictions due to noise. As shown in Figure 5.24, the predictions were less noisy and aligned more closely with the actual performance values, even when the model was built with different sample sets, indicating that the model captures the underlying performance behavior more accurately when noise is reduced. However, future work could further explore how different distributions of Gaussian noise, as well as other types of noise, impact prediction accuracy.

The choice of the sampling strategy influences model construction and, consequently, prediction accuracy, as it determines which configurations are included in the sample sets according to specific criteria. To account for this, we used random sampling as a baseline approach, since it makes no additional assumptions about the configuration space and is widely used in practice. However, other strategies, such as t-wise covering, could lead to different results. In particular, for smaller sample sizes like t1, random sampling may fail to cover all individual configuration options, potentially resulting in higher errors than a coverage-based approach. Even though sample size t1 is large enough to cover all individual options in principle, only a coverage-based strategy would guarantee this, leading to more

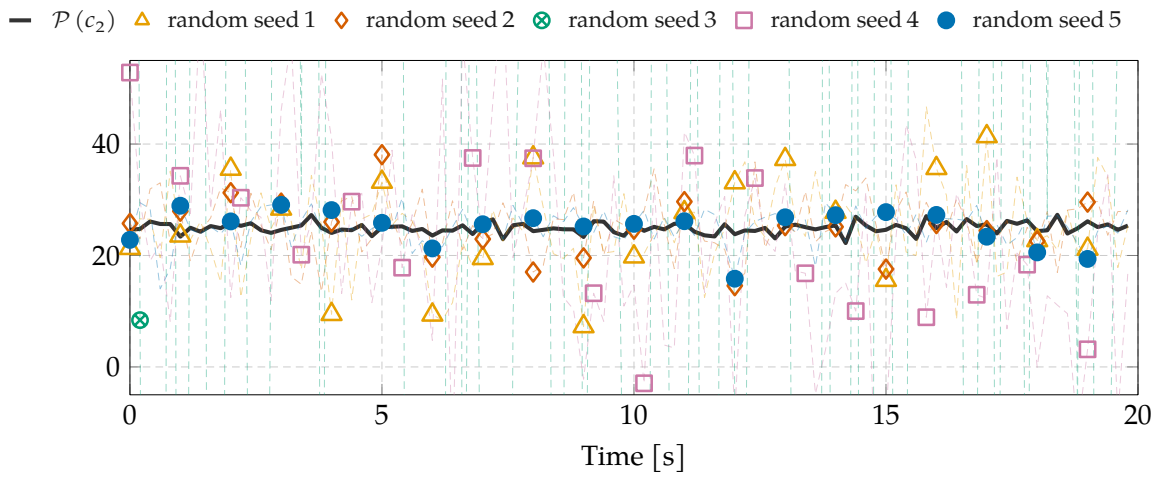


Figure 5.23: The plot displays results for LLVM for configuration c_2, which was unknown for model μ_{t2}. The black line — shows the actual performance behavior of configurations c_2, while the colored markers (△, ◇, ⊗, □, ●) represent the predicted performance from five models μ_{t2} constructed with added noise and five different random seeds.

accurate models. The same applies for sample size t2, where a coverage-based strategy would ensure that all pairs of configuration options are observed, whereas random sampling might leave some pairs uncovered, again potentially increasing errors, especially when interaction influences are present. However, for each sample size, the choice of the sample set is also important. A sample set may be unrepresentative, for instance containing configurations with few distinct configuration options and interactions or unusually high noise. In Figure 5.23 and Figure 5.24, we present predictions for model μ_{t2} from five different sample sets using the same measurement data. These predictions range from cases where the model exhibits severe overfitting (⊗) to cases where predictions are closely aligned with the true performance behavior (●). This illustrates that the choice of the sample set can strongly affect the predictions.

Building on the observed variability in predictions due to different sample sets, another threat arises from the number of sample sets used to construct different models. Ideally, multiple sample sets should be generated for each sample size, with models built from each set and the results aggregated to provide a more robust estimate of prediction accuracy. However, for each sample size, we generated only one sample set and built the corresponding model using that single set. It is possible that this sample set was unrepresentative, for instance containing configurations with few distinct interaction influences or unusually high noise. In the experiments with added noise (c.f. Figure 5.14 and Figure 5.15), we observed that predictions for model μ_{t2} exhibit large fluctuations, which could be attributed either to overfitting to noise or to a bad choice of sampled configurations with particularly high noise. To address this threat, we generated five additional sample sets for sample size t2 in the scenario with 50% of configuration options assigned influences, no interaction influences, and added noise for LLVM and configuration c_2. In this scenario, as discussed in the evaluation, Figure 5.14 shows that the predictions for μ_{t2} were highly noisy and exhibited large errors.

In Figure 5.23, the predictions for models μ_{t2} built with different sample sets across five random seeds are shown. It can be observed that the predictions vary considerably depending

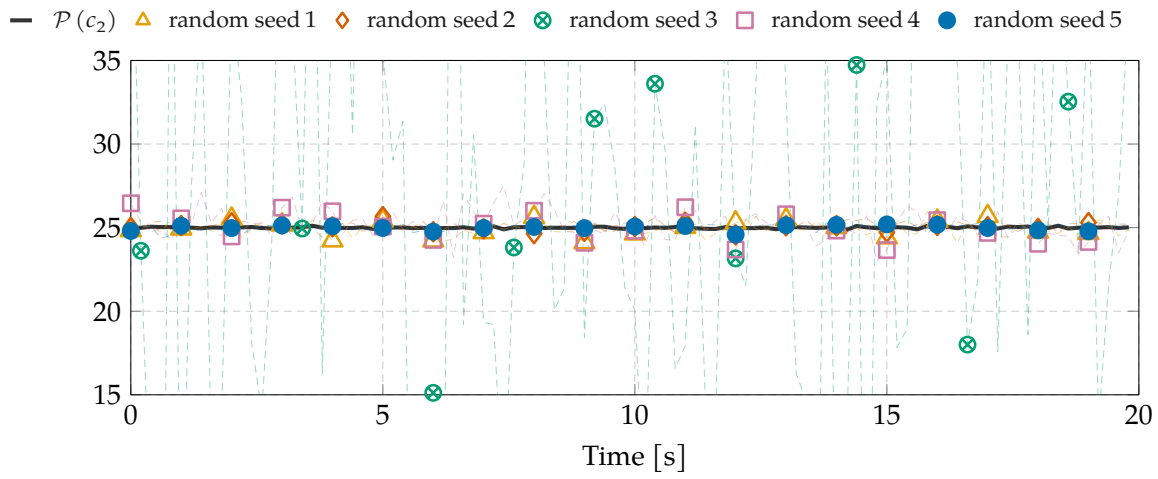


Figure 5.24: The plot displays results for LLVM for configuration c_2, which was unknown for model μ_{t2}. The black line — shows the actual performance behavior of configurations c_2, while the colored markers (△, ◇, ⊗, □, ●) represent the predicted performance from five models μ_{t2} constructed under reduced noise (standard deviation 0.05) and five different random seeds.

on the seed. For the third random seed (⊗), only a single predicted point is visible in the graph, showing that the predictions deviate substantially and indicating that the model overfitted to noise and failed to capture the underlying performance behavior. When model μ_{t2} is built with sample sets for seeds one (△) and four (□), predictions still fluctuate with a value of 20 from the actual performance values, showing amplification of noise, though less extreme than for seed three. Predictions for μ_{t2} constructed with the sample set for seed two (◇) are closer to the actual constant behavior but still exhibit some fluctuations, while predictions for seed five (●) align most closely with the actual values, showing minor deviations.

These observations indicate that the choice of the sample set has a significant impact on the predictions of models μ_{t2}, ranging from severe overfitting to predictions closely aligned with the true performance behavior. This variability likely arises because some configurations in the sample sets were assigned particularly high noise, which the model interprets as part of the performance behavior, resulting in predictions distorted by noise. However, since the predictions still capture the overall trend of the underlying performance function, models μ_{t2} appear to cover a sufficient number of configurations to capture most configuration options and interactions with influences, yet it is still insufficient to regularize the noise. In contrast to models μ_{t1}, which also overfit to noise but exhibit fewer fluctuations, models μ_{t2} may have already seen too many noisy configurations to interpret the noise as part of the performance behavior, leading to predictions distorted by noise.

The strong distortions in predictions for models μ_{t2} could also be explained by the substantial noise added to the performance functions. Since we used Gaussian noise with a large standard deviation of 1.0, we need to examine whether these extreme fluctuations were mainly caused by the amount of noise. To this end, we rebuilt the models μ_{t2} with the same five additional sample sets but reduce the Gaussian noise to a standard deviation of 0.05. The results are shown in Figure 5.24.

With lower noise, the predictions for μ_{t2} across all sample sets are much closer to the actual measurements and no longer exhibit the extreme fluctuations as observed before. Predictions for the third seed ( ) still deviate the most from the actual performance values, but the deviations are much smaller than with higher noise. This supports the conclusion from Figure 5.23 that the fluctuations in the model's prediction for this sample set are caused by overfitting to noise, possibly because configurations in the sample set received high noise. This supports the earlier thought that model for this sample size is overfitting to noise might be the reason for this very bad behavior, because maybe in this sample set there are configurations that got much noise.

For all other seeds, the predictions are very close to the actual performance, with only minor deviations for seeds one ( ), two ( ), and four ( ). For seed five ( ), the predictions remain the closest, aligning almost perfectly with the actual values.

These results indicate that the amount of noise strongly affects the predictions of models μ_{t2}. Higher noise leads to pronounced distortions, as the model seems to learn the noise itself as part of the performance behavior. Nevertheless, even with lower noise, the choice of the sample set still influences accuracy, as reflected in the variability of the predictions across different random seeds.

6 Limitations and Future Work

While our approach demonstrates promising results for predicting performance behavior over time, several limitations need to be considered, and future work is needed to validate and enhance its applicability in real-world scenarios.

An underlying assumption of this work is that each configuration option and their interactions have distinct influences on the overall performance, and that for any configuration, these influences sum up to determine its performance behavior. Our experimental data were constructed according to this assumption, with influences explicitly assigned to configuration options and interactions, so that the behavior of each configuration reflects the sum of its present influences. To validate whether this assumption holds in real-world systems, future work should apply the approach to actual performance measurements. If the model still makes accurate predictions on real-world measurements, this would indicate that performance can be decomposed into influences of individual configuration options and their interactions, and that the models can capture these effects outside of a controlled setting. However, if the predictions deviate substantially or are only accurate for configurations included in the sample set, this would suggest that the model is likely overfitting to seen configurations and that the configuration options may not have a consistent influence on the overall performance.

Extending the evaluation to real-world configurable software systems and their actual performance measurements would enable an assessment of the approach's practical applicability and accuracy in less controlled environments. Such an extension would naturally introduce noise to the performance data, which is inherent in real-world settings. So far, we have only considered Gaussian noise from a single distribution, whereas noise in real-world measurements may exhibit different characteristics, including heavier fluctuations and different distributions. Since small sample sizes already showed signs of amplified noise, a challenge will therefore be to ensure that the learned models remain well-regularized and do not overfit to noisy data, especially when trained on smaller sample sizes.

Another limitation of our approach lies in the use of seeded performance functions, which allowed us to take performance measurements at equidistant time intervals by evaluating the functions at fixed points. In practice, this assumption does not hold, as real-world measurements are taken during execution and are only approximately equidistant due to measurement inaccuracies. To address this issue we need to interpolate the measurements to obtain truly equidistant time intervals before applying the Discrete Fourier Transform.

Moreover, we restricted our study to configurations with equal execution times to ensure that all predictions operate on the same frequency spectrum, which enabled us to attribute influences to specific frequencies. In real-world scenarios, however, execution times frequently vary across configurations, making this assumption unrealistic. Future work could therefore investigate methods for handling unequal execution times and evaluate their effect on

prediction accuracy. Possible approaches include aligning measurements to a common time grid before applying the Discrete Fourier Transform, or pruning performance measurements to a shared execution time. However, the latter is only feasible if execution times across configurations are relatively similar, as otherwise truncating to shorter execution times could risk losing important performance behavior from other configurations with longer execution times.

Finally, our current approach only accounts for performance influences of pairwise interactions between configuration options. In practice, higher-order interactions involving three or more options may also have significant impacts on performance. It remains unclear how well our models would perform when such higher-order interactions are present. Future work could address this by either allowing the optimization solver to directly consider higher-order influences, or by starting from individual configuration options and iteratively building up to higher-order interactions by combining lower-order interactions that have been assigned influences until the prediction error is sufficiently low.

7

Related Work

Understanding and predicting the performance of configurable software systems has been a long standing research direction. One research area is performance modeling of configurable software systems, which also includes sampling and machine learning techniques to model the performance of large configuration spaces [33, 36, 41]. Further research works on performance evolution across different versions or under dynamic conditions [7, 19, 25, 29] and hardware influences which examine how performance changes due to different hardware components [3, 16]. Beyond research specific to configurable systems, we also consider work in time series analysis and forecasting, where the Fourier Transform has been widely adopted to anticipate performance trends and predict future changes [9, 22].

Sampling and Learning Techniques. Software systems are often highly configurable, leading to a configuration space that can reach exponential size [41]. Exploring the entire configuration space to analyze performance behavior is therefore infeasible. Performance modeling addresses this challenge by measuring a subset of all configurations [33] to predict the performance of the entire configuration space. The predictions can be obtained using machine learning techniques [12, 23], which are applied for performance prediction and optimization of configurable software systems. The selection of an appropriate sampling strategy and machine learning technique is an important decision when addressing performance modeling. Previous work has shown that the choice of the sampling strategy has significant influence on the accuracy of the machine learning algorithm [12, 18].

Performance Modeling. While standard machine learning techniques can predict performance, the resulting models are often difficult to interpret. To address this challenge Siegmund et al. [35, 36] proposed performance influence models to understand and predict how configuration options and their interactions affect the performance of software systems. Performance influence models are interpretable models that allow explanations of performance behavior across large configuration spaces. An alternative approach for performance modeling is presented by Guo et al. [14]. By combining several machine learning techniques they build a data-efficient model, which reduces the number of measurements while preserving model accuracy. Epifani et al. [8] explore how adaptive performance models that continuously update their parameters at runtime based on newly collected data from the executing system. In addition to these approaches, numerous other methods for constructing performance models have been proposed [6, 13, 31].

Performance Evolution. The techniques discussed so far focus on analyzing performance within a single release of a system. In contrast, Kaltenecker et al. [19] studied the evolution of

performance behavior across different system releases. They investigated the relationship between configurability and performance evolution, showing that almost every release displays performance changes in some configurations. Although performance changes occur frequently between releases, the affected subset of the configuration space and the responsible configuration options are a relatively small subset.

Similar to the analysis of performance evolution across system releases by Kaltenecker et al. [19], research in embedded systems addresses the performance evolution across system changes. Here, timing analysis techniques are used to assess the impact of updates on the execution time of embedded software [7]. Rather than reevaluating the entire codebase after each update, which is time and resource consuming, differential timimg analysis focuses only on the code segments affected by an update and determines their impact on the execution time.

Hardware Influences. While updates and different system versions represent intentional modifications to a system, performance variability can also arise from the underlying hardware. Berry et al. [3] studied modern computer microprocessors and observed highly variable performance during program execution, demonstrating that performance can be influenced by uncontrollable low-level environmental factors. Antoniou et al. [1] further investigate how differences in client-side hardware contribute to performance variations. Even when executing the same application, performance can vary depending on the hardware of the client. Consequently, understanding the performance of an application requires analyzing the interactions between software and hardware components. Hauswirth et al. [16] address this by collecting time series data from multiple layers of the system such as the operating system, virtual machine and application. For each layer, different metrics are collected and represented as individual time series. This enables reasoning about how different system layers influence overall performance behavior.

System Layer Influences. In addition to layered architectures, many systems are distributed across multiple machines. In distributed systems, a single request can trigger numerous sub-requests across several software components, which makes identifying performance inefficiencies challenging. Zhao et al. [44] developed a profiling tool capable of reconstructing the execution flow of requests across these components. This approach enables the identification of root causes and the specific components responsible for performance issues.

Dynamic Environments and Self-Adapting Systems. Beyond internal system components like hardware and software, performance can also be affected by dynamic external conditions, such as fluctuating workloads and network conditions [11]. Software systems frequently operate under uncertain and time-varying conditions, which can impact their performance [11, 25, 40]. In the area of self-adapting systems, configurable systems can adjust to changing environmental and internal conditions such as time-varying workloads. These systems learn from previous adaptations and uses this knowledge to react to future environmental changes [29, 42].

Time Series Modeling and Forecasting. Besides learning to react to changes in the environment, systems can benefit from anticipating future performance changes. The area of

time series modeling and forecasting is the process of predicting future values based on historical data collected over a certain period of time. Historical data is used to learn from past observations and to identify patterns and trends, enabling predictions of how performance will change over time [9].

Previous work applies time series modeling techniques to enable automatic adjustment of performance thresholds under varying workloads [17]. These thresholds are predefined limits that indicate deviations from expected performance behavior. They are used to detect performance anomalies and trigger alerts when the system misbehaves. Setting these performance thresholds manually is often inflexible and fails to adapt to changing environments. Instead, this approach allows the system to automatically set suitable thresholds and detect performance issues [17].

The Fourier transform is an established algorithm in the area of time-series analysis. Typically, the Fourier analysis is used to model the time series data in the frequency domain. Time series forcasting faces challenges in achieving precision and interpretibility, especially when considering multivariate scenarios. Using the Fourier transform provides interpretability and simplifies the analysis of time series by decomposing them into their frequency representation. Therefore, it supports in identifying and understanding periodic patterns and relationships within historical data [21, 22].

Practical applications have demonstrated these advantages. Prior work [10] applies the Discrete Fourier Transform to student access data from Moodle[1] log files in order to predict the students final course results. The study demonstrates that transforming the time series data into the frequency domain using the DFT preserves the information. Furthermore, since data extracted from log files can be very extensive, the DFT enables effective data compression. By retaining only $5 - 10\%$ of the most significant Fourier coefficients, data size is reduced while improving model accuracy.

1 https://moodle.de

8

Concluding Remarks

Today, most software systems are configurable, resulting in vast configuration spaces [41]. While this flexibility enables software to be tailored to diverse requirements, it also complicates performance prediction because each configuration can exhibit distinct performance behavior. Measuring and analyzing all possible configurations is typically infeasible due to the exponential growth of the configuration space. To predict the performance behavior of a specific configuration, classical performance modeling approaches summarize the performance of configurations as scalar values [35].

In this thesis, we extend classical performance modeling by presenting an approach for predicting the performance behavior of configurable software systems over time. That means, our approach enables us to construct models that output performance sequences over execution time for any given configuration. To build such models, we leverage the Discrete Fourier Transform to map performance traces into their frequency representation. This transformation enables us to formulate the model construction as an optimization problem based on the observed frequency values. We assign influences to configuration options and their interactions such that the difference between predicted and actual frequency values is minimized.

We evaluate the accuracy of our models in an initial experiment. There, we use the configuration spaces of two real-world configurable systems and manually seed functions to their configuration options and their interactions. This setup provides ground-truth performance sequences for configurations in a controlled setting. To explore different scenarios, we vary both the number and the choice of configuration options that get influences assigned, as well as the type of influence functions. For each scenario, we construct models from different sample sets of configurations, taking performance measurements for each configuration either without or with added noise. We evaluate accuracy qualitatively by comparing predicted performance sequences of selected configurations to their ground-truth sequences, and quantitatively by measuring prediction errors across all configurations. For quantitative evaluation, we summarize the difference between predicted and actual sequences by using the mean Integral Squared Error.

We observe that prediction accuracy improves with increasing sample size, with larger sample sizes yielding negligible prediction errors. Smaller sample sizes are mostly able to capture general performance trends but may miss specific influences. When noise is added, small sample sizes tend to overfit on noise, treating noise as part of the underlying behavior. Larger sample sizes are better at regularizing the noise and achieving highly accurate predictions. Overall, we present a methodology for constructing models that enable time-inhomogeneous performance predictions in configurable software systems, with our results demonstrating the feasibility of our models, making our approach a promising direction for further research on real-world systems.

A

Appendix

In the following we show the distribution of prediction errors for the models built in all experiments and for both systems, LLVM (left) and lrzip (right). First, we illustrate results for experiments without synthetic noise, followed by those with synthetic noise. For each case, we distinguish scenarios where influences are assigned to one-fifth, half, or all configuration options. We show results for all three sample sizes and both random seeds used for the function assignments. Within each boxplot, we distinguish between models constructed with seeded behavior where influences are assigned only to individual configuration options (Interaction Degree 1) and those where their interactions are assigned influences (Interaction Degree 2).

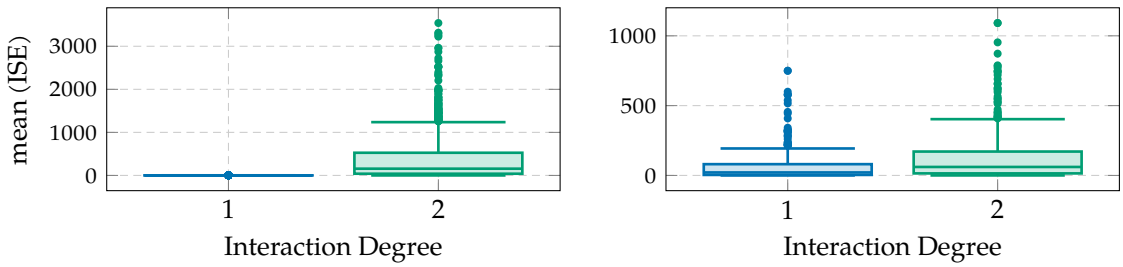
A.1 Boxplots for Measurements without Synthetic Noise

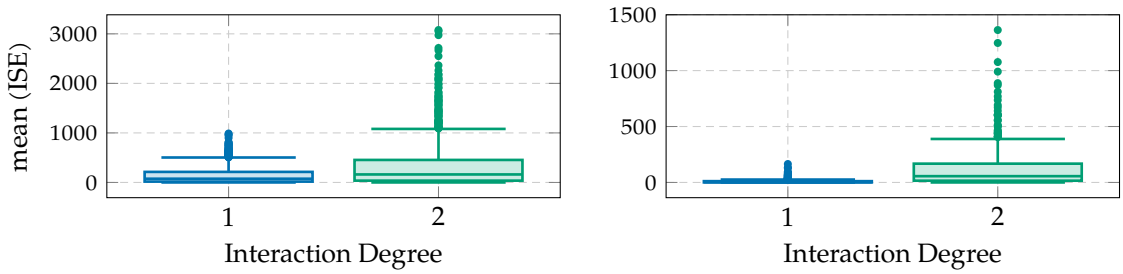
The distributions of models built in experiments without synthetic noise are presented below. Two plots are shown for each of the models μ_{t1}, μ_{t2}, and μ_{t3}, corresponding to different seeds used for selecting configuration options and assigning functions.

Assignment of Influences to One-Fifth of Configuration Options

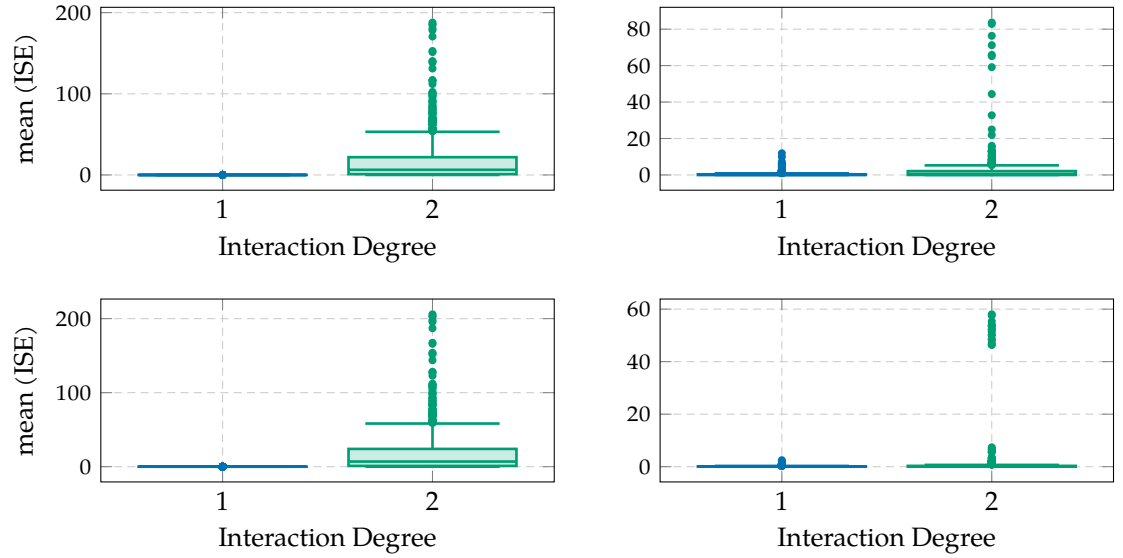
The following presents the results of the experiments in which one-fifth of the configuration options are assigned influences.

Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t1}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.

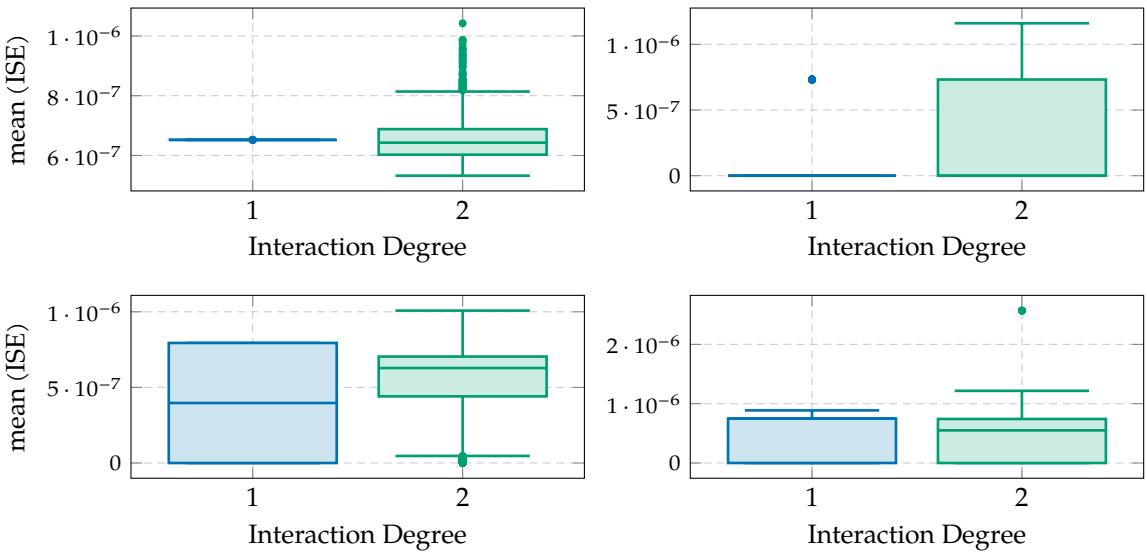




Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t2}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



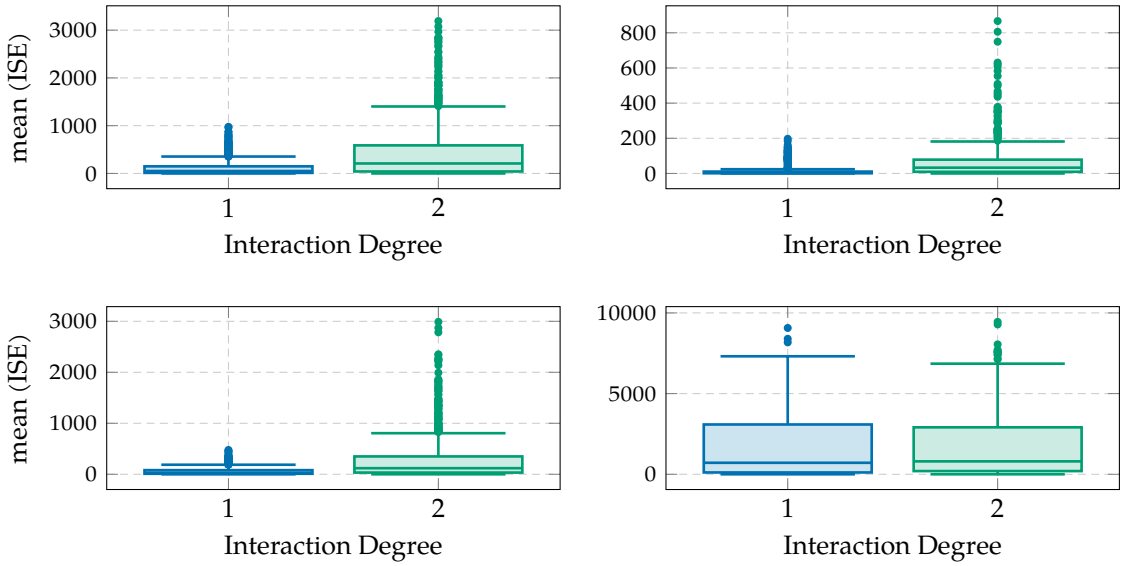
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t3}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



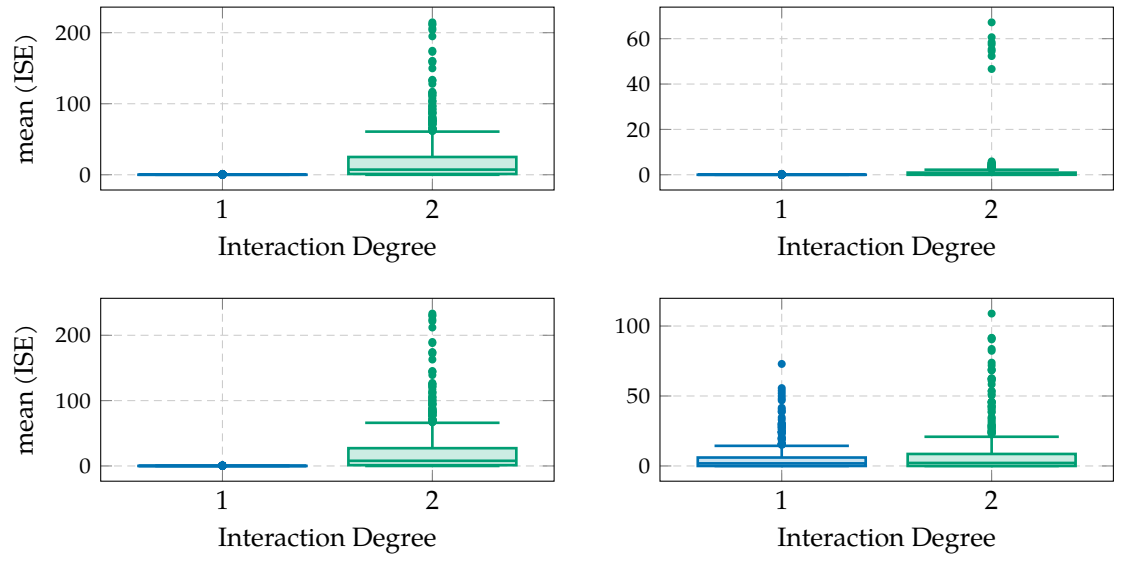
Assignment of Influences to Half of Configuration Options

The following presents the results of the experiments in which half of the configuration options are assigned influences.

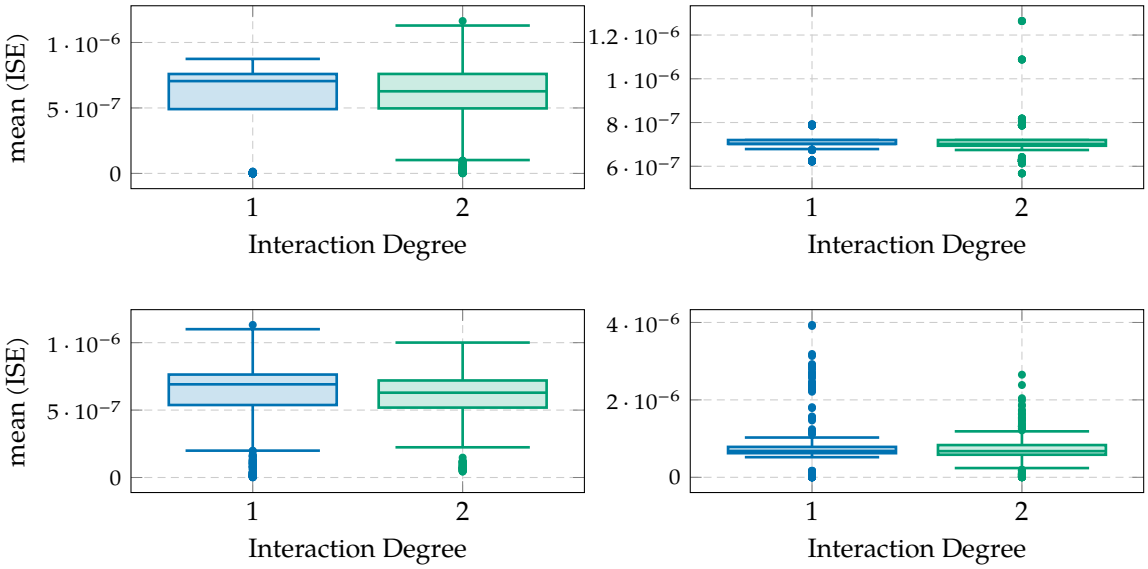
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t1}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t2}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



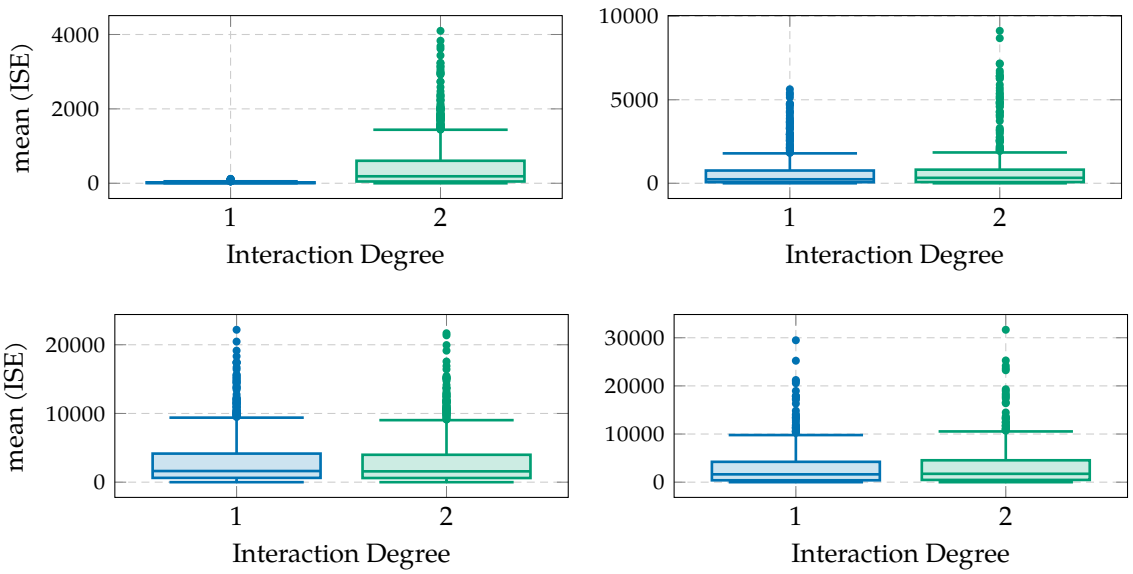
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t3}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



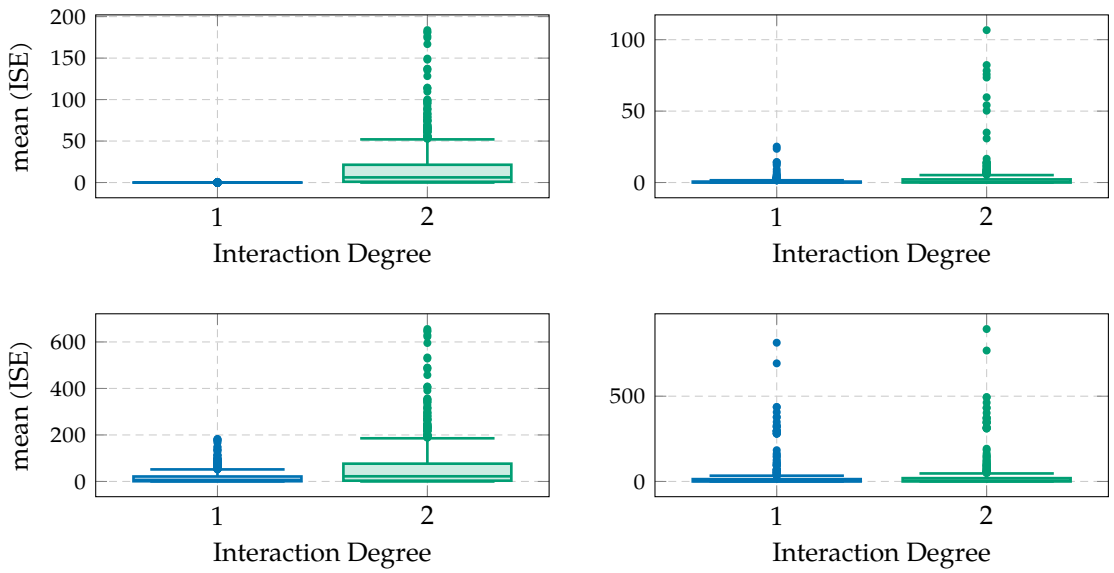
Assignment of Influences to All Configuration Options

The following presents the results of the experiments in which all configuration options are assigned influences.

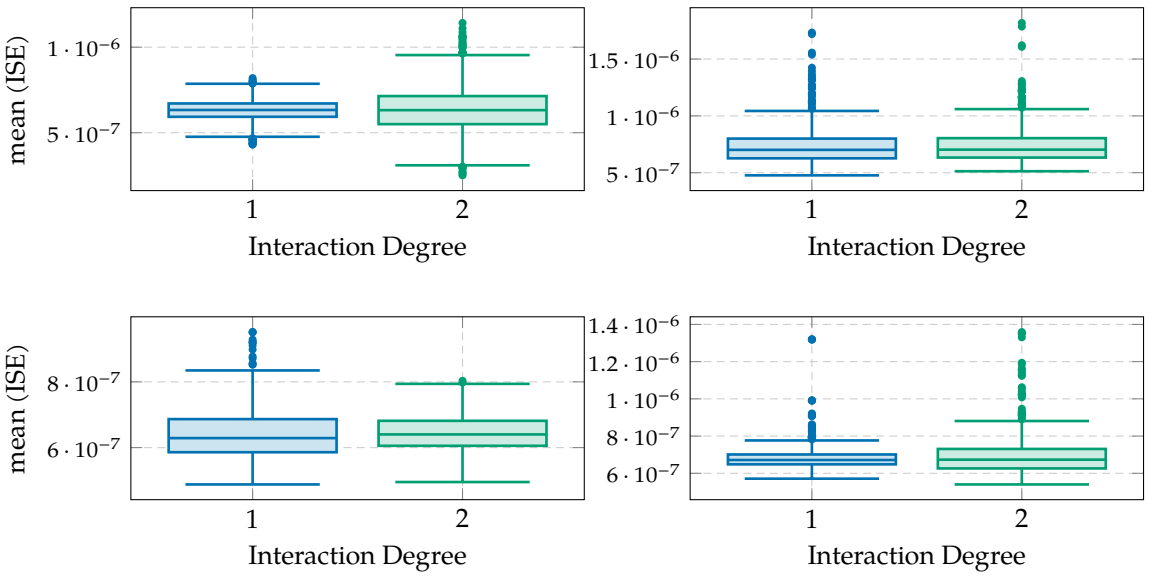
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t1}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t2}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t3}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



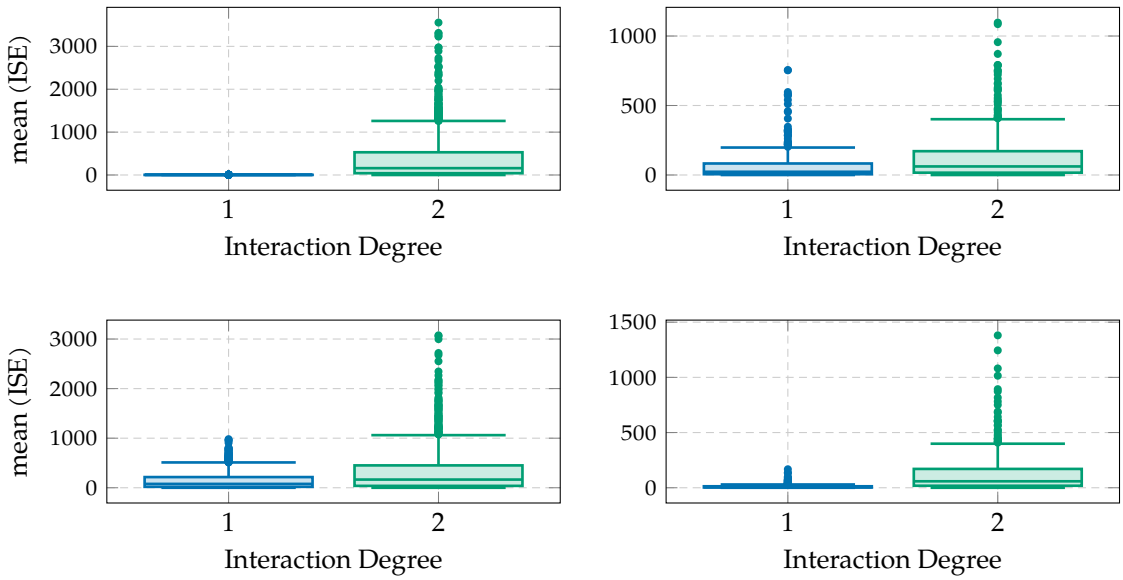
A.2 Boxplots for Measurements with Synthetic Noise

The distributions of models built in experiments without synthetic noise are presented. Two plots are shown for each of the models μ_{t1}, μ_{t2}, and μ_{t3}, corresponding to different seeds used for selecting configuration options and assigning functions.

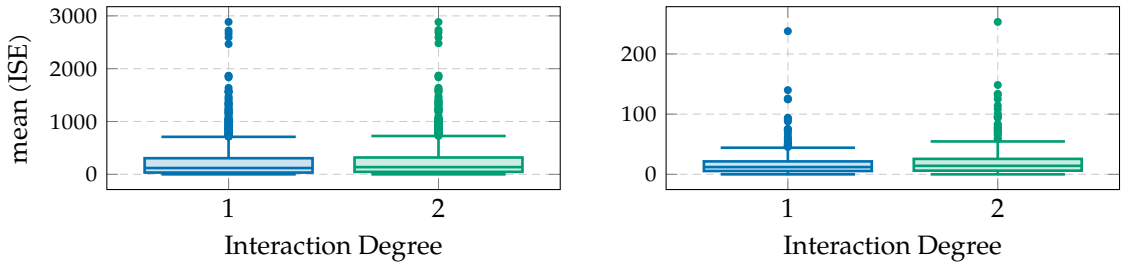
Assignment of Influences to One-Fifth of Configuration Options

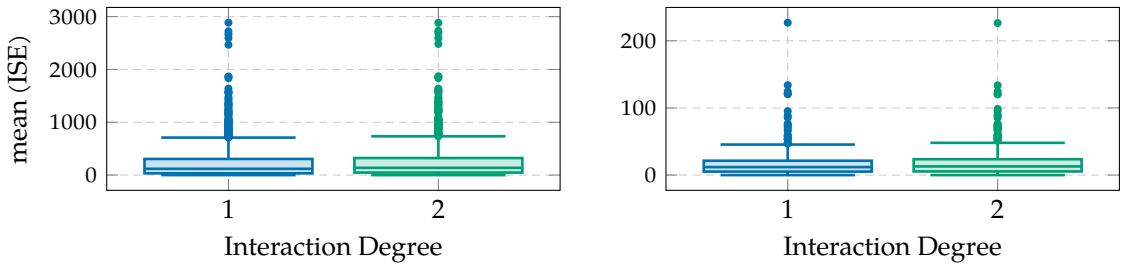
The following presents the results of the experiments in which one-fifth of the configuration options are assigned influences.

Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t1}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.

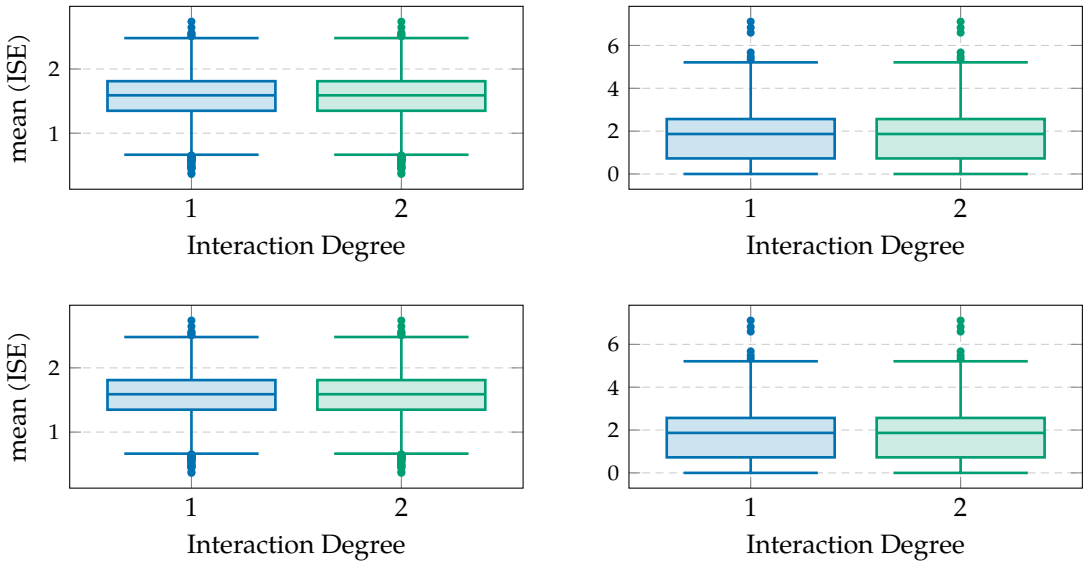


Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t2}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.





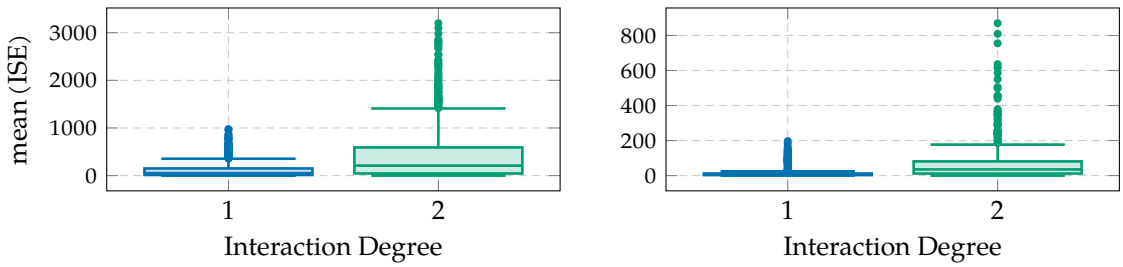
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t3}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.

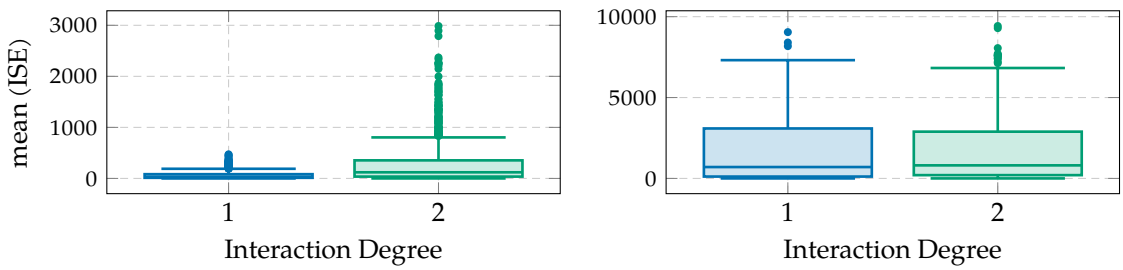


Assignment of Influences to Half of Configuration Options

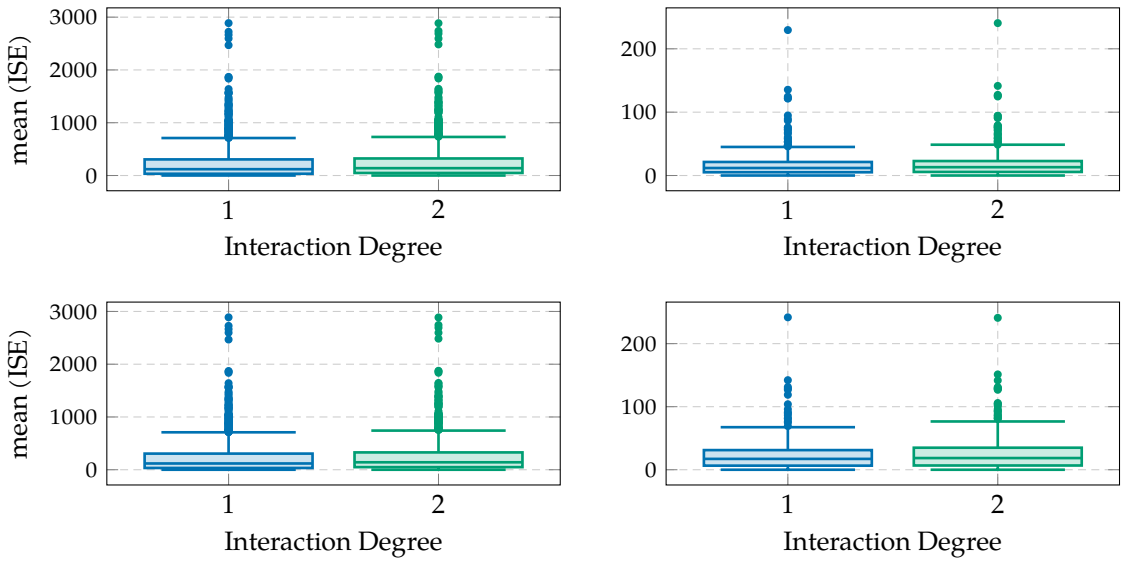
The following presents the results of the experiments in which half of the configuration options are assigned influences.

Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t1}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.

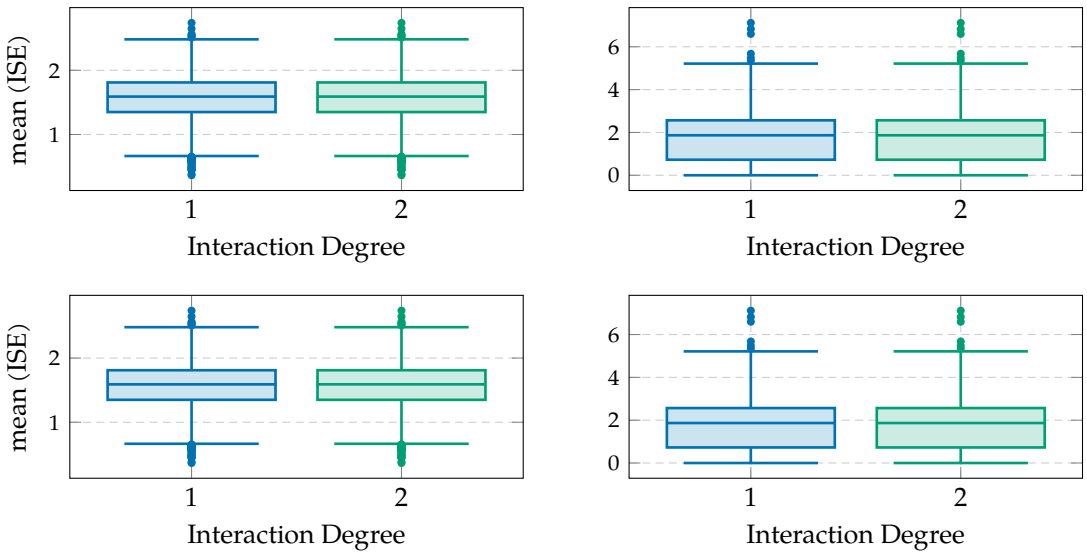




Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t2}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



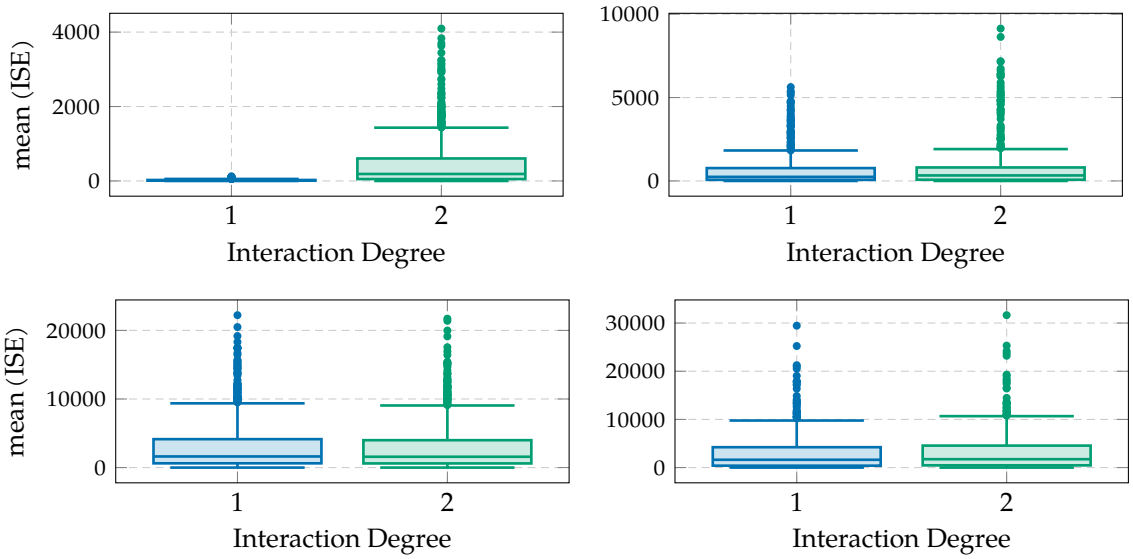
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t3}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



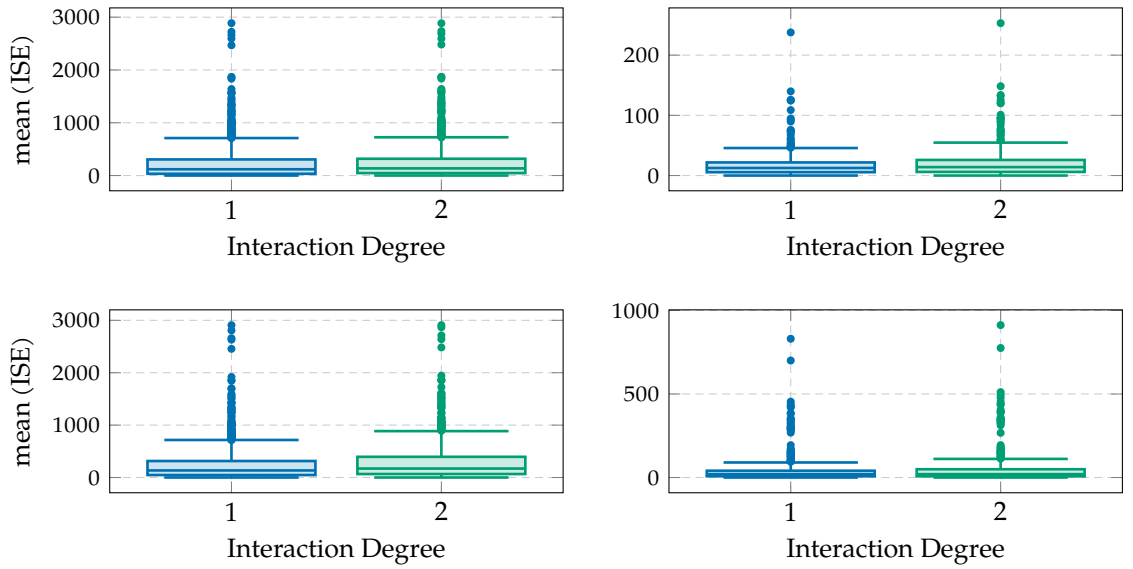
Assignment of Influences to All Configuration Options

The following presents the results of the experiments in which all configuration options are assigned influences.

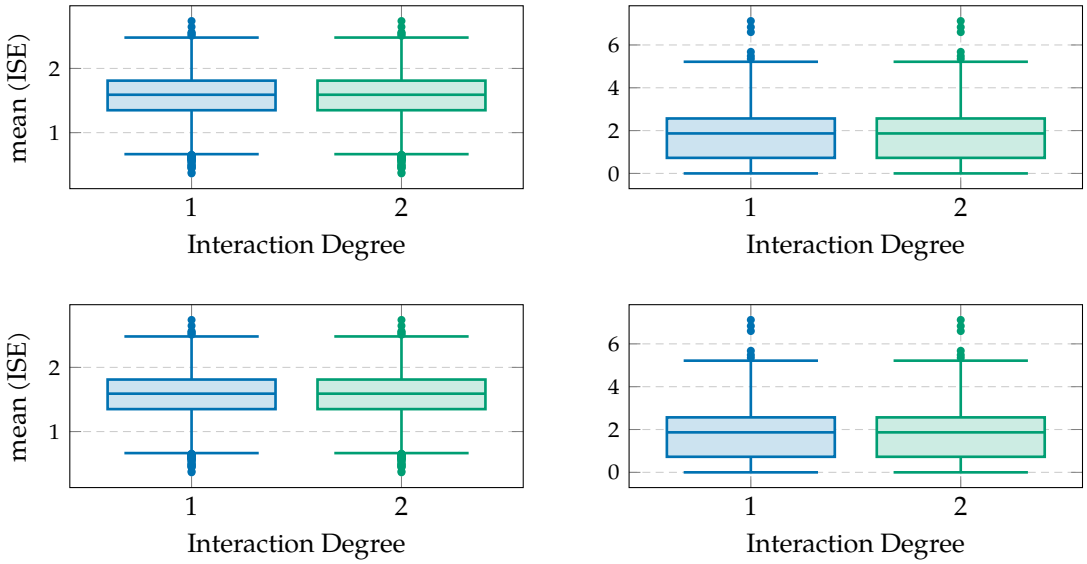
Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t1}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t2}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.



Boxplots for LLVM (left) and lrzip (right) showing the distribution of prediction errors for model μ_{t3}, with the first and second plots showing two different seeds used for selecting configuration options and assigning functions.


mean (ISE)
2
1
1
2
Interaction Degree
6
4
2
0
1
2
Interaction Degree

mean (ISE)
2
1
1
2
Interaction Degree
6
4
2
0
1
2
Interaction Degree

Statement on the Usage of Generative Digital Assistants

For this thesis, the following generative digital assistants have been used: I have used ChatGPT[1] for collecting possible inspiration, text revision and text rewriting. It was used to improve the grammar and sentence structure of my writing and to check for mistakes like comma placements and spelling errors.

In addition, I have used Copilot[2] for code completion. Specifically, it provided suggestions for running the existing evaluation scripts sequentially and, in other parts of the code, for completing repetitive patterns.

I am aware of the potential dangers of using these tools and have used them sensibly with caution and with critical thinking. The core ideas, structure, and content of this thesis were developed independently without relying on these tools. I did not trust the output blindly, but reviewed the suggestions and only applied them when they aligned with my own understanding.

1 https://chat.openai.com/
2 https://github.com/features/copilot

Bibliography

[1] Georgia Antoniou, Haris Volos, and Yiannakis Sazeides. "Taming Performance Variability caused by Client-Side Hardware Configuration." In: *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2024, pp. 94–107.

[2] Sven Apel and Christian Kästner. "An overview of feature-oriented software development." In: *J. Object Technol.* 8.5 (2009), pp. 49–84.

[3] Hugues Berry, Daniel Gracia Pérez, and Olivier Temam. "Chaos in computer performance." In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 16.1 (2006).

[4] Muffy Calder, Mario Kolberg, Evan H Magill, and Stephan Reiff-Marganiec. "Feature interaction: a critical review and considered forecast." In: *Computer Networks* 41.1 (2003), pp. 115–141.

[5] Denis Chemla, Jacques Young, Fabio Badilini, Pierre Maison-Blanche, Hélene Affres, Yves Lecarpentier, and Philippe Chanson. "Comparison of fast Fourier transform and autoregressive spectral analysis for the study of heart rate variability in diabetic patients." In: *International journal of cardiology* 104.3 (2005), pp. 307–313.

[6] Jiezhu Cheng, Cuiyun Gao, and Zibin Zheng. "Hinnperf: Hierarchical interaction neural network for performance prediction of configurable systems." In: *ACM Transactions on Software Engineering and Methodology* 32.2 (2023), pp. 1–30.

[7] Ahmed El Yaacoub, Luca Mottola, Thiemo Voigt, and Philipp Rümmer. "Timing analysis of embedded software updates." In: *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE. 2023, pp. 1–11.

[8] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. "Model evolution by run-time parameter adaptation." In: *2009 IEEE 31st International Conference on Software Engineering*. IEEE. 2009, pp. 111–121.

[9] Valentin Flunkert, David Salinas, and Jan Gasthaus. "Deepar: Probabilistic forecasting with autoregressive recurrent networks." In: *arXiv preprint arXiv:1704.04110* 23 (2017).

[10] Jasna Gamulin, Ozren Gamulin, and Dragutin Kermek. "Using Fourier coefficients in time series analysis for student performance prediction in blended learning environments." In: *Expert systems* 33.2 (2016), pp. 189–200.

[11] Jesús García-Galán, Liliana Pasquale, George Grispos, and Bashar Nuseibeh. "Towards adaptive compliance." In: *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2016, pp. 108–114.

[12] Alexander Grebhahn, Norbert Siegmund, and Sven Apel. "Predicting performance of software configurations: There is no silver bullet." In: *arXiv preprint arXiv:1911.12643* (2019).

[13] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. "Variability-aware performance prediction: A statistical learning approach." In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2013, pp. 301–311.

[14] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. "Data-efficient performance learning for configurable systems." In: *Empirical Software Engineering* 23.3 (2018), pp. 1826–1867.

[15] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. "Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack." In: *Empirical Software Engineering* 24.2 (2019), pp. 674–717.

[16] Matthias Hauswirth, Peter F Sweeney, and Amer Diwan. "Temporal vertical profiling." In: *Software: Practice and Experience* 40.8 (2010), pp. 627–654.

[17] Peter Hoogenboom and Jay Lepreau. "Computer System Performance Problem Detection Using Time Series Model." In: *USENIX Summer*. 1993, pp. 15–32.

[18] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. "The interplay of sampling and machine learning for software performance prediction." In: *IEEE Software* 37.4 (2020), pp. 58–66.

[19] Christian Kaltenecker, Stefan Mühlbauer, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. "Performance evolution of configurable software systems: an empirical study." In: *Empirical Software Engineering* 28.6 (2023), p. 152.

[20] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. *Feature-oriented domain analysis (FODA) feasibility study*. Tech. rep. 1990.

[21] Minjung Kim, Yusuke Hioka, and Michael Witbrock. "Neural Fourier Modelling: A Highly Compact Approach to Time-Series Analysis." In: *arXiv preprint arXiv:2410.04703* (2024).

[22] Noam Koren and Kira Radinsky. "Interpretable multivariate time series forecasting using neural fourier transform." In: *arXiv preprint arXiv:2405.13812* (2024).

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), pp. 436–444.

[24] Nathan Lenssen. "Applications of Fourier Analysis to Audio Signal Processing: An Investigation of Chord Detection Algorithms." In: (2013).

[25] Luc Lesoil, Mathieu Acher, Xhevahire Tërnava, Arnaud Blouin, and Jean-Marc Jézéquel. "The interplay of compile-time and run-time options for performance prediction." In: *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A*. 2021, pp. 100–111.

[26] Maria C Mariani, Hector Gonzalez-Huizar, Md Al Masum Bhuiyan, and Osei K Tweneboah. "Using dynamic Fourier analysis to discriminate between seismic signals from natural earthquakes and mining explosions." In: *aims geosciences* 3.3 (2017), pp. 438–449.

[27] Armstrong Nhlabatsi, Robin Laney, and Bashar Nuseibeh. "Feature interaction: The security threat from within software systems." In: *Progress in Informatics* 5.75 (2008), p. 1.

[28] SS Pandey, Manu Pratap Singh, and Vikas Pandey. "Image transformation and compression using Fourier transformation." In: *Int. J. Curr. Eng. Technol* 5.2 (2015), pp. 1178–1182.

[29] Juliane Päßler, Maurice H ter Beek, Ferruccio Damiani, Einar Broch Johnsen, and Silvia Lizeth Tapia Tarifa. "Analysing self-adaptive systems as software product lines." In: *Proceedings of the 2025 29th ACM International Systems and Software Product Line Conference-Volume A*. 2025, pp. 237–237.

[30] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. "Learning software configuration spaces: A systematic literature review." In: *Journal of Systems and Software* 182 (2021), p. 111044.

[31] Ilia Pietri, Gideon Juve, Ewa Deelman, and Rizos Sakellariou. "A performance model to estimate execution time of scientific workflows on the cloud." In: *2014 9th workshop on workflows in support of large-scale science*. IEEE. 2014, pp. 11–19.

[32] Mohammed H Rasheed, Omar M Salih, Mohammed M Siddeq, and Marcos A Rodrigues. "Image compression based on 2D Discrete Fourier Transform and matrix minimization algorithm." In: *Array* 6 (2020), p. 100024.

[33] Atrisha Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. "Cost-efficient sampling for performance prediction of configurable systems (t)." In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2015, pp. 342–352.

[34] Ervin Sejdić, Igor Djurović, and LJubiša Stanković. "Fractional Fourier transform as a signal processing tool: An overview of recent developments." In: *Signal Processing* 91.6 (2011), pp. 1351–1369.

[35] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. "Performance-influence models for highly configurable systems." In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015, pp. 284–294.

[36] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. "Predicting performance via automated feature-interaction detection." In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, pp. 167–177.

[37] Norbert Siegmund, Marko Rosenmuller, Christian Kastner, Paolo G Giarrusso, Sven Apel, and Sergiy S Kolesnikov. "Scalable prediction of non-functional properties in software product lines." In: *2011 15th International Software Product Line Conference*. IEEE. 2011, pp. 160–169.

[38] S Simón-Díaz and A Herrero. "Fourier method of determining the rotational velocities in OB stars." In: *Astronomy & Astrophysics* 468.3 (2007), pp. 1063–1073.

[39] Julio Sincero, Wolfgang Schröder-Preikschat, and Olaf Spinczyk. "Approaching non-functional properties of software product lines: Learning from products." In: *2010 Asia Pacific Software Engineering Conference*. IEEE. 2010, pp. 147–155.

[40] Danny Weyns, Radu Calinescu, Raffaela Mirandola, Kenji Tei, Maribel Acosta, Nelly Bencomo, Amel Bennaceur, Nicolas Boltz, Tomas Bures, Javier Camara, et al. "Towards a research agenda for understanding and managing uncertainty in self-adaptive systems." In: *ACM SIGSOFT Software Engineering Notes* 48.4 (2023), pp. 20–36.

[41] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. "Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software." In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015, pp. 307–319.

[42] Yulong Ye, Tao Chen, and Miqing Li. "Distilled lifelong self-adaptation for configurable systems." In: *arXiv preprint arXiv:2501.00840* (2025).

[43] Sai Zhang and Michael D Ernst. "Which configuration option should I change?" In: *Proceedings of the 36th international conference on software engineering*. 2014, pp. 152–163.

[44] Xu Zhao, Kirk Rodrigues, Yu Luo, Ding Yuan, and Michael Stumm. "{Non-Intrusive} performance profiling for entire software stacks based on the flow reconstruction principle." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 603–618.

[45] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. "Bestconfig: tapping the performance potential of systems via automatic configuration tuning." In: *Proceedings of the 2017 symposium on cloud computing*. 2017, pp. 338–350.