



Tools and benchmarks evolve: what is their impact on parameter tuning in SBSE experiments?

Amid Golmohammadi¹ · Man Zhang² · Andrea Arcuri^{1,3}

Received: 21 August 2024 / Accepted: 15 September 2025
© The Author(s) 2025

Abstract

In this article, we explore the impact of tool development and its evolution in Search-Based Software Engineering (SBSE) research. As a research tool evolves throughout the years, experiments with novel techniques might require reevaluation of previous studies, especially regarding parameter tuning. These reevaluations also give the opportunity to address the threats to external validity of these previous studies by employing a larger selection of artifacts. To conduct the replicated experiments in this study, the search-based fuzzer EvoMASTER is chosen. This SBSE tool has been developed and extended throughout several years (since 2016) and tens of scientific studies. Among the chosen tool's parameters, 6 were carefully selected based on 5 previous studies that we replicate in this article with the latest version of EvoMASTER. The replication is applied across an expanded set of artifacts compared to the original replicated studies. Our objective is to validate the robustness and validity of previous findings and to determine the need for parameter tuning in response to the tool's continuous development. Beyond replication, we explored parameter tuning by testing 729 different configurations to find a more performant parameter set, which is later validated through additional rounds of experiments. Additionally, we analyzed the impact of individual parameters on test generation performance using machine learning models, providing insights into their relative effects. Our findings indicate that, although most parameters maintain their efficacy, 2 of them require adjustment. Furthermore, the investigation into the effects of combining different parameter values reveals that carefully optimized configurations can outperform default settings. These findings highlight the importance of regularly reevaluating parameter settings to enhance tool performance in SBSE research.

Keywords SBSE · Parameter tuning · Replication study · SBST · REST APIs · White-box test generation

Communicated by: Justyna Petke.

Extended author information available on the last page of the article

1 Introduction

A search algorithm can involve many parameters that need to be configured. For instance, when utilizing a genetic algorithm (Holland 1992), it is required to provide several parameter values, such as population size, crossover probability, selection mechanism (e.g., roulette wheel or tournament based), and mutation probability. These parameter choices can have a significant impact on the effectiveness of the search algorithm. In the worst case scenario, an unsuitable choice of parameters is likely to make it nearly impossible to solve the given problem. Therefore, it is necessary to tune the parameters in a way that maximizes the search algorithm's performance (Arcuri and Fraser 2013).

Many Search-Based Software Engineering (SBSE) tools, such as EvoSuite (Fraser and Arcuri 2011) and EvoMASTER (Arcuri et al. 2021), come with default parameters that were found to work reasonably well in past experiments. However, as tools evolve, they incorporate newer methods. This raises the question of whether earlier studies, including their parameter tuning, remain valid. For instance, if research finds a new method X performs best at setting x' , would this conclusion hold if years later a new method Y is subsequently introduced with optimal value y ? Or would a different x'' be a better choice when $Y = y$? Essentially, our objective is to determine whether prolonged development of a tool, often used with default settings from past studies, would demand adjustments to these default settings.

In this article, we aim to find out if the optimal values found by the original studies remain effective, despite the ongoing evolution of SBSE tools. If so, it would also be important to investigate whether those findings apply to a larger set of experimental subjects, as they were derived from empirical evaluation on a limited set of subjects. Apart from trying to find an optimal value for each parameter in isolation, we aim to study the impact of combining them together to check if a parameter set providing better results can be found for this problem domain. In other words, the goal is to determine whether there is a single set of parameter values that, while not necessarily the best for each individual experimental subject, achieves the highest overall performance on average across all examined subjects. This would make it a viable configuration for the SBSE tool to use as the new, improved “default” setting.

For attaining these research objectives, any open-source SBSE tool that has been developed throughout several years could be used. For our empirical experiments, we chose EvoMASTER (Arcuri et al. 2021), as it fits these requirements. Another popular tool that could have been used is EvoSuite (Fraser and Arcuri 2011). We chose EvoMASTER out of convenience, as we are its maintainers, and any new found optimization is integrated in its latest release, benefiting all the developers in industry who download it. On the other hand, at the time of writing, EvoSuite has not got any new release in years (latest version 1.2.0 is from 2021).¹

EvoMASTER is a tool engineered for automatic generation of system-level test cases for REST APIs (Arcuri 2019), as well as GraphQL (Belhadi et al. 2023) and Remote Procedure Call (RPC) (Zhang et al. 2023). It uses SBSE techniques, such as the Many Independent Objective (MIO) algorithm (Arcuri 2017) to optimize test cases for maximizing code coverage and fault detection.

¹ <https://github.com/EvoSuite/evosuite/releases>

To address these research challenges, numerous parameters and studies could be replicated. Nevertheless, we selected 6 parameters, as re-tuning every parameter would not be feasible given the computational cost. These parameters were chosen based on a review of 5 foundational EvoMASTER studies from 2018 to 2021 (Arcuri 2018; Arcuri and Galeotti 2020; Zhang and Arcuri 2021a, b; Arcuri and Galeotti 2021).

We used EvoMASTER 1.6.2-SNAPSHOT for our experiments, which was the latest version at the time of writing this article. We selected 14 different open-source REST APIs from the EMB repository (EvoMaster Benchmark (EMB) 2025) for our case studies, including those in the original studies we replicate. To increase our study's relevance to real-world scenarios, we also included one industrial REST API.

A set of specific configuration values was assigned to each parameter taken from the previous studies (Arcuri 2018; Arcuri and Galeotti 2020; Zhang and Arcuri 2021a, b; Arcuri and Galeotti 2021). The outcomes of these configurations were assessed using three key metrics: line coverage, branch coverage, and fault detection. Then, an extensive empirical analysis was conducted, which could have taken more than two and a half years (922 days) if not run in parallel. This involved finding the parameter configuration for each of the 15 REST APIs with the highest line coverage, branch coverage, and found faults. This analysis included the verification of whether previously set default values are outperformed by a new value. The results help us understand if previous research findings are still valid. Such research enriches our understanding of SBSE, potentially highlighting existing gaps in handling complex tools with numerous settings.

Our research progresses through five meticulously structured empirical phases to assess and validate the impact of parameter tuning in an evolving SBSE context. Initially, we assessed the impact of each of the 6 parameters individually, employing different settings (3 values per configuration, including the default setting) across the set of 15 Systems Under Test (SUT), which are the experimental subjects in this study. To further refine our findings, a second set of experiments was conducted, where we expanded the parameter space, except for one parameter of type `Enum` for which this was not applicable, by testing a finer granularity of values. This allowed us to assess whether previous conclusions remained valid when a broader range of values was considered. Following this, an extensive combination of these parameters, totaling $3^6 = 729$ distinct configurations, was evaluated to discover the most effective configuration. However, as each of these 10935 (i.e., 729 configurations \times 15 SUTs) experiments takes 1 hour to run, only one experiment repetition was carried out. In the fourth phase, the study rigorously examined the identified optimal settings against the default configuration in order to validate their superiority and to explore the existence of a “best” configuration. A larger number of repeated experiments with different random seeds was carried out to take into account the possible side-effects of only using one repetition in the previous phase. Finally, the fifth round undertook a comprehensive evaluation of the previously identified optimal configuration by applying a significantly higher time budget. Unlike the one-hour time budget used in the initial four rounds, this phase utilized a ten-hour budget for a more in-depth assessment. This final phase aimed to confirm the optimized setting’s efficacy across different SUTs under more extensive testing conditions.

This study highlights the necessity for ongoing updates to parameter settings as tools evolve. It demonstrated that, although some previous parameter recommendations remain effective, others require adjustments to align with advancements in tool capabilities. Moreover, our investigation into the combination of different parameter settings shows a sig-

nificant improvement in test generation performance by resulting in higher values for our three chosen metrics (i.e., line coverage, branch coverage, and fault detection). We found `probOfApplySQLActionToCreateResources` to be the most influential on fault detection. It was also revealed that `feedbackDirectedSampling` was the most important one for line and branch coverage. This was done by employing machine learning models that have been evaluated based on their Mean Squared Error (MSE) and Coefficient of Determination (R^2) (Hyndman and Koehler 2006; Steel et al. 1960). For each metric, one model that had the best evaluation scores was picked. Furthermore, comparing optimized settings against the default setting across several SUTs proved the strength of the configuration which yields the highest number of faults detected. This analysis not only broadens our knowledge of the evolving field of SBSE, but also prepares the ground for future studies focused on enhancing testing approaches through careful parameter adjustments and updates to reflect tool advancements.

In several aspects, this study can be considered as a form of “*replication study*”. Existing experiments published in the literature are replicated, under two different variables: (1) a larger set of SUTs, and (2) a more mature version of the employed tool. Note that the first author of this article, who did this replication study, was not involved in any of these previous studies that are now replicated. Unfortunately, albeit their importance (Juristo and Gómez 2012), replication studies are rare in the literature (Da Silva et al. 2014).

It is not in the scope of this article to propose any *novel* way to do parameter tuning. Our goal is to provide sound empirical evidence to answer our research questions about tuning in the context of *evolving* SBSE tools. As far as we know, this is the first study of this kind in the literature of SBSE.

While SBSE research covers a wide range of software engineering tasks (Harman and Jones 2001; Harman et al. 2012), studies specifically investigating the impact of parameter tuning through replication and empirical evaluation remain limited. As for all replication and tuning studies in the SBSE literature (e.g., Arcuri and Fraser 2013; Sayyad et al. 2013; Kotelyanskii and Kapfhammer 2014; Zamani and Hemmati 2020; Tawosi et al. 2021, as discussed in more detail in Section 2), only one tool (e.g., EvoSuite) and one SBSE problem (e.g., unit test generation) has been empirically studied. Using more tools and more SBSE problem instances could of course lead to more general results. However, it would significantly increase the complexity and size of the study, beyond the typical scope and constraints (e.g., in terms of page size) of an article publication. For example, an in-depth empirical analysis as done in this paper (for EvoMASTER on REST APIs) for a second tool in a different domain would likely require a similar amount of space and time to carry out.

The thousands of practitioners in industry that have downloaded and use EvoMASTER do directly benefit from a study like this, as the newly found optimized values have now been set in the latest releases of EvoMASTER. For researchers, this study provides sound empirical evidence on the importance of re-tuning tools that are developed and extended throughout the years. Each time a novel SBSE technique is proposed, it could have side-effects on previously set configurations. This is not just a theoretical possibility, but, as shown in this article, it has real-world practical consequences. This article provides sound empirical evidence to support such a claim. Large re-tuning parameter experiments are expensive, but, as shown in this article, they can pay off significantly. But, of course, how much impact they would have on other tools and problem domains would be up to investigation.

This article is an extension of Golmohammadi et al. (2023), in which three previous works (Arcuri 2018; Zhang and Arcuri 2021a, b) were replicated. In the current article, this previous work is extended by adding another two studies (Arcuri and Galeotti 2020, 2021) to be replicated. These two additional parameters are `maxSQLInitActionsPerMissingData` and `baseTaintAnalysisProbability`. Their tuning results are discussed in Sections 5.4.2 and 5.4.6, respectively. We also provide a study with a more fine-grained set of values for the experimented parameters (Section 5.5). In the previous study (Golmohammadi et al. 2023), the selected parameters were studied separately. However, in this study, apart from adding another two parameters from the newly added articles, we aimed to study how different settings of the 6 parameters will perform when combined together (Sections 5.6 and 5.8), as well as using machine-learning techniques to study their relations (Section 5.7). Furthermore, we also studied the impact of running fuzzing sessions for longer (Section 5.9). This approach enables us to find an optimal combination of the settings and determine the impact of each parameter on the performance of the test generation.

This article is structured as follows. Section 2 discusses related work. The parameters chosen to be tuned are introduced and discussed in Section 3. Section 4 gives an overview of the employed machine learning algorithms and the metrics to evaluate them. Section 5 includes the details of the five rounds of the empirical study, along with the findings from these studies. Section 6 provides our reflections on the results obtained in our empirical study. Possible threats to validity are discussed in Section 7. Finally, Section 8 concludes the article and discusses future work.

2 Related Work

Parameter tuning is crucial for the design and effectiveness of Evolutionary Algorithms (EAs). The selection of parameter values can significantly affect the performance of the EA, so much so that an EA with optimally chosen parameters can perform better than one with suboptimal parameters (Eiben and Smit 2011). Recent research also emphasizes the benefits of dynamic parameter control, where parameter values are adjusted during the optimization process to better match the current state of the algorithm. This approach, which includes state-dependent, success-based, and learning-inspired mechanisms, has been shown to significantly improve algorithm performance over static parameter tuning (Doerr and Doerr 2020).

The research conducted by Freisleben and Härtfelder (1993) offers a technique for identifying the most effective genetic algorithm (GA) for a specific issue by framing it as an optimization problem and utilizing another GA to resolve it. The approach is structured in two layers for optimization: the lower layer employs a subordinate GA that manipulates gene sequences symbolizing potential answers, while the upper layer features a main GA that manages a set of these subordinate GAs, each encoded as its own gene sequence. Every subordinate GA functions autonomously to generate a solution, and its performance has an impact on the activity of the main GA. The generational cycles in both layers function independently of one another, and the ultimate goal of the primary GA is to identify the gene sequence with the highest fitness value as the optimal GA for the original problem.

Several studies have delved into the effects of parameter adjustments on the efficacy of search-based test generation methods. One such study by Arcuri and Fraser (2013) involved

more than a million trials of generating unit tests for object-oriented software through the EvoSuite tool. The findings suggest that, although parameter adjustments can influence the performance of search algorithms, identifying settings that markedly exceed the default values is challenging. This implies that sticking with default settings may be a practical choice, considering the time and effort involved in fine-tuning parameters.

The study by Arcuri and Fraser (2013) has been replicated by other researchers. Sayyad et al. (2013) investigated the effects of parameter tuning on the performance of the two algorithms, Indicator-Based Evolutionary Algorithm (IBEA), and Non-Dominated Sorting Genetic Algorithm (NSGA-II) when applied to the problem of configuring Software Product Lines (SPLs) in the presence of stakeholder preferences including reliability and cost. The outcomes confirmed the findings in the original study (Arcuri and Fraser 2013).

Another replicated study (Kotelyanskii and Kapfhammer 2014) by Kotelyanskii and Kapfhammer provided additional evidence for the findings in Arcuri and Fraser (2013). The results reinforce previous findings, indicating that the EvoSuite configuration identified by the well-known Sequential Parameter Optimization Toolbox (SPOT) did not notably surpass default settings, aligning with Response Surface Methodology (RSM) that yielded no benefit in Arcuri and Fraser (2013). While this outcome is not favorable, it adds to the body of evidence highlighting the difficulty of optimizing a sophisticated search-based test data generator. Additionally, the study suggests that the default parameters of EvoSuite, presumably established by field experts, are appropriate for both future research endeavors and practical testing applications.

In another study (Zamani and Hemmati 2020), Zamani and Hemmati introduced an innovative measure called “Tuning Gain” to assess the cost-effectiveness of fine-tuning specific classes. They recommend predicting this metric by examining the static attributes of the source code classes. The evaluation of their approach entailed an exhaustive analysis of 1,200 hyper-parameter configurations of EvoSuite for 250 classes across 19 projects. The analysis employed a Meta-GA tuning approach, comparing the effectiveness of EvoSuite with and against a proposed class prioritization method. Based on these predicted Tuning Gains, classes are ranked and prioritized for fine-tuning. When resources for tuning are limited, the budget is allocated to a selected group of classes that are expected to yield the highest Tuning Gain, resulting in a tenfold improvement in overall tuning results.

Parameter tuning can provide significantly better results, but only on a specific problem domain. It is theoretically unfeasible to design an algorithm, or tune an existing one, that performs optimally on all possible problem domains, as stated by the No Free Lunch theorem (Wolpert and Macready 1997). While the theorem relies on assumptions, such as uniform distributions over all objective functions, that do not typically hold in realistic SBSE settings, it still highlights that the best parameters found for a specific SBSE tool is unlikely to generalize to other domains, even within other SBSE problems.

Outside of the tuning sphere, the necessity of study replication in search-based software engineering has been examined. Tawosi et al. (2021) focused on revisiting and augmenting an existing study concerning CoGEE, an advanced tool for multi-objective software effort estimation. Using a separate implementation and a reliable baseline known as LP4EE, the study’s internal and external validity were strengthened. The performance and efficacy of CoGEE were further validated by testing it against four additional evolutionary algorithms and the JMetal Java framework. The findings confirmed not only CoGEE’s superior performance, but also its algorithm-agnostic nature.

Another article (Črepinšek et al. 2014) addressed the complexities of replicating computational studies in applied evolutionary computing. This study highlighted frequent shortcomings in the documentation of existing research, which hampers replication efforts. It also emphasized the significance of adhering to rigorous scientific and statistical standards, particularly given the stochastic behavior of evolutionary algorithms. The primary aim was to offer guidelines that help mitigate common errors, thereby enhancing the reproducibility and quality of future research in the field.

To the best of our knowledge, no existing literature addresses the unique challenge we explore in this research. These challenges include trying to study the impact of parameter tuning on the previously set configurations after new parameters are introduced as the result of tool evolution, finding the most impactful parameters on the test generation performance, and studying the impact of the scale of case studies on SBSE research. We aim to adjust certain carefully chosen parameters of a search-based tool, designed to generate system-level test cases specifically for REST APIs. It is desired to find out if the previous findings are still valid.

3 Considered Parameters

Parameters settings play an important role in search algorithms by adjusting factors such as the scope of the search space, convergence rate, and execution time. Inadequate or unoptimized parameters can result in poor performance, whereas carefully chosen settings have the potential to substantially boost the quality of the outcomes. Therefore, fine-tuning these parameters is crucial for achieving optimal algorithmic results (Golmohammadi et al. 2023).

The *Many Independent Objective (MIO)* (Arcuri 2017, 2018) algorithm is an evolutionary algorithm designed specifically for generating test suites. This algorithm is used not only in EvoMASTER for test case generation, but also integrated as an option in other tools such as *EvoSuite* (Fraser and Arcuri 2011) and *Pynguin* (Lukasczyk and Fraser 2022). MIO's success and efficiency are influenced by several parameters.

In this section, we elaborate on the six parameters of EvoMASTER selected for our research: `feedbackDirectedSampling`, `maxSQLInitActionsPerMissingData`, `probOfApplySQLActionToCreateResources`, `probOfRandomSampling`, `focusedSearchActivationTime`, `baseTaintAnalysisProbability`. Each of these parameters will be explained in more detail in the next subsections.

While EvoMASTER 1.6.2-SNAPSHOT provides 189 different parameters² at the time of writing this article, many do not influence the search process, such as `outputFolder` which merely designates the directory path for saving generated test classes. Moreover, we have not included any of the EvoMASTER's 67 Boolean parameters, i.e., the ones that activate or deactivate a whole feature like `enableTrackIndividual` that is for enabling tracking the history of modifications of the individuals during the search. Therefore, these Boolean parameters were excluded as there is minimal room for their adjustment. Consequently, out of 79 parameters that impact the search process and are not of type Boolean, we intentionally narrowed our focus to these six key parameters that are either of numeric type (i.e., `Int` and `Double`) or `Enum` which is a type that defines a set of named constants.

²For the interested reader, those parameters can be found in the EMConfig.kt file in EvoMASTER (EvoMaster 2025)

Our selection was influenced by insights gained from five significant previous studies (Arcuri 2018; Arcuri and Galeotti 2020; Zhang and Arcuri 2021a, b; Arcuri and Galeotti 2021) in which EvoMASTER was developed with new key techniques, i.e., `feedbackDirectedSampling` referring to a strategy for prioritizing objectives during search, `maxSQLInitActionsPerMissingData` and `probOfApplySQLActionToCreateResources` referring to a strategy for initializing data for system-level test case generation for the SUT interacted with databases, `probOfRandomSampling` and `focusedSearchActivationTime` referring to strategies for exploring and exploiting search space, and `baseTaintAnalysisProbability` referring to a strategy for generating test data using taint analysis. Detailed information about the 6 selected parameters is shown in Table 1., including their data type, range, default value and the last year they were changed. These pieces of information are inferred by inspecting the Git history of `EMConfig.kt` file under the EvoMASTER 's repository on GitHub (EvoMaster 2025).

3.1 Feedback-Directed Sampling

In the MIO algorithm, Feedback-Directed Sampling (FDS) is a technique that steers test input creation using insights from earlier tests. This approach focuses on areas of the search space that are more likely to yield higher quality test cases which is more optimal than exploring the search space randomly. This can be particularly helpful in scenarios with a limited search budget. This feedback mechanism aids in guiding the evolutionary process to ensure that the population evolves towards specific goals, such as detecting faults or optimizing coverage. This strategy also aids in preventing the unnecessary expenditure of time on unattainable goals, as these would lack a gradient (Golmohammadi et al. 2023).

In EvoMASTER, the `feedbackDirectedSampling` parameter, which is of type `Enum`, controls this process. The original study (Arcuri 2018) examined all three possible values: `NONE`, `LAST`, and `FOCUSED`. The `LAST` value was chosen as the new default for EvoMASTER. Based on the Git history of `EMConfig.kt` file under EvoMASTER 's repository, this default setting has remained unchanged since 2017 (Golmohammadi et al. 2023).

3.2 Max SQL Init Actions per Missing Data

This parameter serves to regulate the extent of data initialization during the test case generation, particularly when the SUT interacts with a database. In such cases, populating the

Table 1. Information about the six selected parameters inferred from the Git history of `EMConfig.kt` file under EvoMASTER 's repository

Parameter	Type	Range	De- fault Value	Last Change
<code>feedbackDirectedSampling</code>	<code>Enum</code>	{ <code>NONE</code> , <code>LAST</code> , <code>FOCUSED</code> }	<code>LAST</code>	2017
<code>maxSQLInitActionsPerMissingData</code>	<code>Int</code>	[1,]	5	2018
<code>probOfApplySQLActionToCreateResources</code>	<code>Double</code>	[0, 1]	0.5	2021
<code>probOfRandomSampling</code>	<code>Double</code>	[0, 1]	0.5	2016
<code>focusedSearchActivationTime</code>	<code>Double</code>	[0, 1]	0.5	2017
<code>baseTaintAnalysisProbability</code>	<code>Double</code>	[0, 1]	0.9	2019

database with appropriate data is critical for effective automated system testing. An empty or incorrectly populated database can result in incomplete or inefficient test outcomes. However, the task of database population is not always straightforward. There are scenarios where the database might be “read-only”, or where the sequence of operations needed to populate the database is too complicated to generate easily.

To address these limitations, Arcuri and Galeotti (2020) proposes integrating SQL heuristics with existing bytecode heuristics. The idea is to capture every SELECT query executed by the SUT and adjust the optimization targets to ensure these queries yield non-empty data sets. This new approach shares similarities with branch distance heuristics (Korel 1990; Alshraideh and Bottaci 2006) but focuses on the WHERE clauses of SQL queries. The SQL heuristics are designed to work collaboratively with the search process, which can involve multiple operations such as HTTP calls and could include a large number of SQL queries in each test case.

The parameter `maxSQLInitActionsPerMissingData` controls the maximum number of new rows that can be generated for each specific SQL SELECT query during the SQL data generation phase for testing. The minimum possible value and the default value of this parameter are 1 and 5, respectively. The default setting has not changed since 2018.

3.3 Probability of Applying SQL Action to Create Resources

Testing REST APIs can be challenging, particularly when developing system-level tests due to the complex interactions they may have with SQL databases. In REST architecture, endpoints usually represent the resources, their associated data, and the various operations that can be conducted upon them. As highlighted by Zhang et al. (2019, 2021), interacting with these resources across all their different states via HTTP endpoints is beneficial for improving code coverage in white-box testing.

Moreover, manipulating resources becomes more straightforward through the direct execution of SQL commands (Zhang and Arcuri 2021a). The parameter named `probOfApplySQLActionToCreateResources` defines the probability of employing such SQL-based manipulations. It is a probabilistic setting with a range between 0.0 and 1.0. The current default value of 0.5 in EvoMASTER remains based on a study from 2021 (Zhang and Arcuri 2021a) and has remained unchanged since then.

3.4 Probability of Random Sampling

The likelihood of implementing random sampling is a fundamental aspect of the MIO algorithm (Arcuri 2017). It is controlled by the `probOfRandomSampling` parameter which is instrumental in striking a balance between *exploration*, which aims to discover new and untried solutions, and *exploitation*, which refines existing successful solutions. Achieving an appropriate balance between these two aspects is essential for the efficacy of the algorithm. Overemphasizing one component could lead to neglecting the other. This could result in either missed opportunities for new solutions or failure to refine existing ones. Placing too much weight on random sampling may reduce the search’s effectiveness, while a value that is too low might impede exploration which increases the risk of entrapment in local optima (Golmohammadi et al. 2023).

In other words, this parameter determines the probability that EvoMASTER will formulate a completely new test case rather than employing existing test cases or other search methodologies. For instance, a `probOfRandomSampling` value of `0.2` suggests a 20% chance that a fresh test case will be generated randomly, without using insights from prior test cases (i.e., the population of test cases maintained by MIO is not utilized for mutation) (Golmohammadi et al. 2023).

While the `probOfRandomSampling` parameter was not specifically studied in the initial MIO articles (Arcuri 2017, 2018), it was explored in subsequent research that extended MIO with adaptive hyper-mutation (Zhang and Arcuri 2021b). Nonetheless, its default value of `0.5`, which was reaffirmed as optimal in Zhang and Arcuri (2021b), has remained unchanged since 2016. As it is a probabilistic parameter, its value can range from `0.0` to `1.0`.

3.5 Focused Search Activation Time

The parameter `focusedSearchActivationTime` in EvoMASTER represents the proportion of the overall search duration after which the software shifts from a broad exploratory approach to a more precise, focused search. For example, a configuration of 20% implies that following the first 20% of the total search time spent in exploratory mode, EvoMASTER will switch to a focused search strategy for the subsequent 80%. It is worth noting that a few parameters, including `probOfRandomSampling`, gradually reduce over time until this phase of focused searching commences. Careful tuning of this parameter is crucial to achieve a balance between extensive exploration and effective, targeted searching (Golmohammadi et al. 2023).

Similar to the `probOfRandomSampling` parameter, the parameter `focusedSearchActivationTime` was not examined in the initial articles about MIO (Arcuri 2017, 2018). This parameter was studied in the research that augmented MIO with adaptive hyper-mutation (Zhang and Arcuri 2021b). It is a probabilistic setting with a range between `0.0` and `1.0`. However, its standard value of `0.5` (which was discovered as optimal in Zhang and Arcuri 2021b) has remained unchanged since 2017.

3.6 Base Taint Analysis Probability

Dynamic taint analysis involves the real-time marking and tracking of specific data during the program execution. It serves a range of applications from information security to software testing and debugging. Especially in the security domain, taint analysis is crucial for determining whether untrusted inputs could compromise sensitive operations. A simplified form of this analysis can be adopted to track input variables in white-box testing of APIs (Arcuri and Galeotti 2021).

The `baseTaintAnalysisProbability` parameter adjusts the probability of using input tracking to inspect how inputs affect the SUT. As mentioned above, this is a probabilistic setting with a range between `0.0` and `1.0`. The default value of `0.9`, which has not been modified since 2019, indicates a high probability (i.e., 90%) that this type of analysis will be applied.

4 Machine Learning Models and Model Evaluation Metrics

Machine learning algorithms can be utilized to analyze the impact of input variables on the output dependent variables. In the context of our research, we aim to leverage them in order to gain insights into the importance of the parameters on the performance of the test case generation. In other words, the idea is to find out which parameters have the highest impact on the metrics. Among the vast number of machine learning algorithms, *Random Forest Regressor* (Breiman 2001), *Gradient Boosting Regressor* (Friedman 2001), *Decision Tree Regressor* (Breiman 2017), and *Linear Regression* (Pearson 1901) were selected for this study. The following brief overview helps to understand how these models contribute to diverse analytical challenges and which metrics are used to evaluate them.

4.1 Machine Learning Models

4.1.1 Random Forest Regressor

Random Forest (Breiman 2001) is an ensemble learning method based on decision tree algorithms that builds many decision trees during training to produce the average prediction from these trees. This method is valued for its strength and flexibility in regression tasks, effectively handling large datasets and high-dimensional spaces. It significantly reduces the chance of overfitting by combining multiple decision trees, leading to more reliable and steady predictions. Breiman's study (Breiman 2001) highlights Random Forest's capability to deal with complex structures in large datasets, making it especially suitable for exploring the complex effects of test generation parameters. Its skill in determining the significance of variables for predicting outcomes highlights its usefulness in different types of data analysis, including both classification and regression, justifying its choice for our study.

4.1.2 Gradient Boosting Regressor

Gradient Boosting (Friedman 2001) is a machine learning technique that constructs a series of models in an incremental fashion, with each subsequent model focusing on correcting the residuals or errors left by the previous models. This iterative process of building models to predict the residuals, combined through a weighted sum or majority vote, allows for the optimization of various loss functions. This makes Gradient Boosting a versatile tool for a broad spectrum of predictive tasks. Its capability to incrementally improve model accuracy by concentrating on difficult-to-predict instances is a significant advantage, particularly in dealing with complex data sets that embody non-linear relationships among variables. The choice of Gradient Boosting for our study is based on its demonstrated effectiveness in reducing both bias and variance, thereby enhancing predictive accuracy across diverse problems. This has been extensively validated in the literature, notably by Friedman in his work on Gradient Boosting Machines, where he elucidates the methodology's efficiency in greedy function approximation (Friedman 2001). This supports our selection of Gradient Boosting Regressor, capitalizing on its ability to address the nuanced challenges presented by our data, including modeling complex, non-linear relationships, thereby justifying its inclusion among the algorithms explored in this study.

4.1.3 Decision Tree Regressor

Decision Trees (Breiman 2017) are a non-parametric supervised learning approach that can be applied to both classification and regression tasks. They aim to predict the target variable's value through simple decision rules derived from data features by dividing the dataset into subsets and simultaneously developing a corresponding decision tree. This process concludes in a tree comprised of decision and leaf nodes which leads to providing a straightforward and interpretable model (Quinlan 1986).

The Decision Tree Regressor (Breiman 2017) is known for its excellent clarity among machine learning algorithms. It shows how different parameters affect outcomes which can give insights into the parameter's effects on test generation metrics. Quinlan's research (Quinlan 1986) highlights the simplicity of decision trees, which makes it easy to see how individual parameters influence results, making it a suitable choice for analyzing the impacts of parameters on test generation metrics.

4.1.4 Linear Regression

Linear Regression (Pearson 1901) is one of the simplest and most widely used statistical techniques for predictive modeling. They are used in various fields from economics to biology for their simplicity and interpretability (Montgomery et al. 2021). Linear Regression is used to model the relationship between a scalar dependent variable and one or more independent variables by fitting a linear equation to observed data. *Simple Linear Regression* is the term used when the model uses a single dependent variable and a single independent variable. When there are multiple variables, it is called *Multiple Linear Regression*. In this paper, we take advantage of Multiple Linear Regression but we simply refer to it as Linear Regression. The key to Linear Regression is the assumption of a linear relationship between the input variables and the output. Its wide application across various fields offers a simple yet powerful tool for modeling relationships and making predictions in diverse research areas (Weisberg 2005).

4.2 Model Evaluation Metrics

4.2.1 Mean Squared Error (MSE)

The MSE (Hastie et al. 2009) is a statistical measure used to quantify the average squared difference between estimated values and the actual values. It serves as a fundamental metric in regression analysis to assess the accuracy of predictive models. The computation of MSE involves squaring the errors (the differences between predicted and observed values) to eliminate the influence of negative signs and then averaging these squared errors over the dataset. This approach ensures that larger errors are given disproportionately more weight, highlighting significant discrepancies between predictions and reality. MSE provides a straightforward, quantitative measure to evaluate model performance, where a lower MSE indicates a closer fit to the data (Hyndman and Koehler 2006).

4.2.2 R² Score

The R² score or Coefficient of Determination (Wooldridge et al. 2016) is the proportion of the total variation of the dependent values that can be explained by the regression model. In other words, it shows how well fitted the regression line is to the data. It usually ranges from 0 to 1, where a score closer to 1 indicates that the model's predictions perfectly match the observed data. In contrast, a score of 0 would mean the model does not explain any of the variability (Chicco et al. 2021). Negative values for R² score are also possible, which indicate that the model performed poorly, though they are not common (Chicco et al. 2021). The R² score is calculated by comparing the fitted model's predictions with the mean of the observed data to provide insight into the model's explanatory power and accuracy in reflecting underlying data patterns.

5 Empirical Study

In the present study, our objectives are to address the subsequent seven research questions:

RQ1 How do the outcomes of the replicated study with the newest version of a SBSE tool correspond to the original findings for each parameter in terms of line coverage, branch coverage, and found faults?

RQ2 How does the addition of more case studies affect the generalizability and reliability of the findings from the initial studies?

RQ3 How does expanding the parameter space impact the validity of previous findings on parameter tuning in SBSE?

RQ4 What is the impact of combining the different values of the parameters together on line coverage, branch coverage, and found faults?

RQ5 Which parameter has the highest impact on the performance of the test generation in terms of the three metrics?

RQ6 Is there a universally “best” parameter configuration for optimizing test generation outcomes?

RQ7 How does the identified “best” parameter configuration perform under a heightened time budget across various SUTs?

RQ1 and RQ2 lay the groundwork for our study by focusing on the important aspects of replication and generalizability. RQ3 builds upon this foundation by investigating whether expanding the parameter space through testing a broader range of values affects the validity of previous findings and leads to different or more refined conclusions on optimal parameter settings. These questions are essential as they help us determine whether the outcomes of prior parameter studies remain consistent with the latest version of the tool and across a more diverse set of SUTs. RQ4 then takes us further by exploring how different combinations of parameters work together, which sheds light on their collective impact on the effectiveness of test generation. RQ5 is aimed at pinpointing the parameters that have the most significant influence on test generation, which could significantly simplify future tuning operations. Lastly, RQ6 and RQ7 are closely related. These two research questions focus on determining whether a universally optimal configuration exists and how robust it is when subjected to a more extended time budget. This step-by-step approach, starting from validation

Table 2 Descriptive statistics of 15 REST APIs in our replicated study

SUT	#SourceFiles	#LOCs	#Endpoints
<i>catwatch</i>	106	9,636	14
<i>cwa-verification</i>	47	3,955	5
<i>features-service</i>	39	2,275	18
<i>genome-nexus</i>	405	30,004	23
<i>gestaohospital</i>	33	3,506	20
<i>ind0</i>	103	17,039	20
<i>languagetool</i>	1,385	174,781	2
<i>market</i>	124	9,861	13
<i>ocvn</i>	526	45,521	258
<i>proxyprint</i>	73	8,338	74
<i>rest-ncs</i>	9	605	6
<i>rest-news</i>	11	857	7
<i>rest-scs</i>	13	862	11
<i>restcountries</i>	24	1,977	22
<i>scout-api</i>	93	9,736	49
Total 15	2,991	318,953	542

ing past findings to identifying an ideal parameter set, forms a thorough methodology for improving the efficiency and effectiveness of parameter tuning in SBSE tools.

5.1 Case Studies

Our experiments were carried out utilizing a pre-existing collection of Web/Enterprise Applications specifically for REST API testing, known as EMB (EvoMaster Benchmark (EMB) 2025; Arcuri et al. 2023). This corpus has appeared in several recent research works (e.g., Arcuri and Galeotti 2021; Kim et al. 2022; Zhang and Arcuri 2023). It is worth mentioning that all of the replicated studies we chose for this experiment (Arcuri 2018; Arcuri and Galeotti 2020; Zhang and Arcuri 2021a, b; Arcuri and Galeotti 2021) also made use of the EMB corpus. Over time, this corpus has expanded to include new Web APIs; it has grown from containing five REST APIs in its initial 2017 release³ to featuring 14 REST APIs in its most recent version 1.6.2-SNAPSHOT in 2023 used in this study.

In the current study, all 14 open-source REST APIs from the EMB corpus, along with an additional REST API from an industrial source was incorporated. This provides a more diverse set of SUTs than those in the original studies that are being replicated (Arcuri 2018; Arcuri and Galeotti 2020; Zhang and Arcuri 2021a, b; Arcuri and Galeotti 2021). We categorized the newly incorporated APIs separately from those used in the original research, labeling our findings as **Original SUTs** and **Other SUTs**. Metrics for each case study, such as the number of source files (#SourceFiles), lines of code (#LOCs), and available endpoints (#Endpoints) for every REST API, are displayed in Table 2.

5.2 Experiment Setting Design

We approached this empirical study through five key steps, represented as five sets of experiments in Table 3. To investigate the impacts of parameters across the long-term evolution of EvoMASTER, we first took a close look at how each parameter individually affected the

³ <https://github.com/EMResearch/EMB/tree/v0.1.1>

Table 3 Purpose of each experiment set, dependencies among them, and how they relate to RQs

Set	Purpose	Depends on	Related RQs
1st	Re-evaluate the individual performance of each parameter supplemented with additional case studies.	—	RQ1, RQ2
2nd	Investigate individual performance of each parameter using a broader range of values than prior works.	—	RQ3
3rd	Study the combined performance of parameters and determine the optimal settings for each metric.	—	RQ4, RQ5
4th	Examine the best configuration per metric from the combined performance study (i.e., the 3rd set) and compare them to the default using additional runs to identify the optimal configuration.	3rd Set	RQ6
5th	Assess the robustness of the best configuration from the 4th set compared to the default under an extended time budget.	4th Set	RQ7

SUTs, re-evaluating the three configurations of each parameter studied in prior works and extending the analysis with additional case studies. Then, to account for potential optimal configurations that may not have been examined in the prior study, we conducted another set of experiments with a finer granularity of parameter values to obtain more precise conclusions.

In the next phase, we focused on examining parameter interactions by evaluating performance across combined configurations. We combined these parameters into 729 different configurations to figure out which ones worked best. Then, we picked the top settings yielded from the third phase for each metric and compared them with the default settings to determine if they truly outperformed and to discover if a single “best” configuration exists.

Finally, we extended our evaluation with a much longer time budget (i.e., 10 hours instead of the 1 hour used earlier) in order to confirm if the best configuration held up across the SUTs under these conditions. More detailed information about each of these steps and how they address the respective RQs are provided later in this section, and parameter configurations for each set of these experiments are presented in Table 4..

The first set of experiments is designed to evaluate how the updated tool version compares with prior results (RQ1), and to assess the impact of including more SUTs on the generalizability of the findings (RQ2). For the first set of experiments, which is aimed to study each parameter separately, with 6 parameters and 3 settings each (one default setting and two other values), we ended up having 13 different settings in total, i.e., $6 \times 2 + 1$. Given 15 REST APIs, each run with a one-hour time budget and 10 repetitions, the total computation time exceeded 81 days, i.e., $13 \times 15 \times 10 \times 1\text{h} \div 24\text{h} = 81\text{d}$. The point of having 10 repetitions was to increase the validity of the results and ensure that the outcomes are not a random occurrence.

In the second set of experiments, we focused on whether extending the parameter space leads to refined or divergent conclusions from previous studies. We conducted experiments to study each parameter with an expanded range of values beyond those in the original studies. We did not include `feedbackDirectedSampling`, as it is of type `Enum` and did not have any additional values in `EvoMASTER` that were omitted. For `maxSQLInitActionsPerMissingData`, we added three more values in this experiment, resulting in a total of five values in addition to the default. For the four other parameters, which were of type `double` and represented probabilities, we increased the number of tested

Table 4 Parameter configurations across the five experiment sets

Parameter	1st Set	2nd Set	3rd Set	4th Set	5th Set
feedbackDirectedSampling	<u>LAST</u> , NONE, FOCUSSED	-	Cartesian product of the configuration sets of all six parameters in the 1st Set	Default and best configurations for and overall code coverage and fault detec- tion from the 3rd con- figura- tions	Default
maxSqlInitActions-PerMissingData	<u>1,3,5</u>	<u>1,3,5,7,15,20</u>			
probOfApplySQLActi-onToCreateResources	<u>0,1,0,3,0,5</u>	<u>0,5,[0,0,0,5] and (0,5,1,0]</u> at 0,1 intervals			
probOfRandomSampling	<u>0,2,0,5,0,8</u>				
focusedSearchActivationTime	<u>0,2,0,5,0,8</u>				
baseTaintAnalysis-Probability	<u>0,1,0,5,0,9</u>				
# of Settings	$1 + 2 \times 6 = 13$	$1 + 5 \times 1 + 10 \times 4 = 46 + 2)^6 = 729$		$1 + 2 = 3$	$1 + 1 = 2$
# of SUTs	All 15 SUTs were applied across all five experiment sets				
Time Budget	1h	1h	1h	1h	10h
# of Repetitions	10	10	1	20	5

Default values are underlined

values from three to eleven, covering the entire range from 0.0 to 1.0 in increments of 0.1 (i.e., 0.0, 0.1, 0.2, ..., 0.9, 1.0). In total, this resulted in 46 different settings to evaluate, i.e., $5 + (4 \times 10) + 1 = 46$. With 15 REST APIs, a one-hour time budget per run, and 10 repetitions, the total computation time amounted to more than 287 days, calculated as $46 \times 15 \times 10 \times 1\text{h} \div 24\text{h} = 287\text{d}$.

For the third set of experiments, which was designed to examine the effects of parameter combinations (RQ4) and identify the most impactful parameter overall (RQ5). To find the best set of parameter settings by combining them, we needed to study $3^6 = 729$ distinct configurations. Running them for 15 SUTs with a one-hour time budget took more than 455 days of computation time, i.e., $729 \times 15 \times 1\text{h} \div 24\text{h} = 455\text{d}$. Since, unlike the first set, we had much more settings to run, it was not practical to have repetitions.

To evaluate whether a consistently superior configuration exists, in the fourth set of experiments, we picked the best parameter settings per metric (i.e., line coverage, branch coverage, and found faults) along with the default setting and ran them for 20 times with a one-hour time budget. As it was not possible to have repetitions in the previous set of experiments (i.e., 3rd set), this set of experiments could help ensure that the achieved best configurations for each metric are actually better than that of default. However, instead of having 4 distinct configurations, we had 3, as the best settings for line coverage and branch coverage were the same in the third set of experiments. This round of experiments required more than 37 days of computation time to run (i.e., $3 \times 15 \times 20 \times 1\text{h} \div 24\text{h} = 37\text{d}$).

In the last set of experiments, we studied the robustness of the overall best configuration under a significantly longer time budget (RQ7). We ran the best configuration from the previous round (i.e., 4th set) along with the default one with a 10-hour time budget 5 times. This was to thoroughly study how the best setting yielded by the previous rounds of experiments performs compared to the default setting. It took more than 62 days of computation time to run, i.e., $2 \times 15 \times 5 \times 10\text{h} \div 24\text{h} = 62\text{d}$.

By establishing a time limit, users set a constraint to ensure EVO^{MASTER} delivers outcomes within the desired amount of time. This involves a delicate balance, as setting a too short time limit might restrict the tool's ability to explore a wide range of test cases, possibly leading to less thorough or effective tests. Conversely, allowing more time gives the evolutionary algorithm additional cycles to refine the test cases, which could enhance the quality of the results. However, this benefit requires a longer wait for the completion of the process. This flexibility enables users to customize how the tool operates to meet their particular requirements and constraints. For this replication study, we decided on a one-hour time budget for the first three sets of experiments, since it is the stopping criterion utilized in the most recent study of EVO^{MASTER} (Zhang and Arcuri 2023). In the fifth set of experiments, given that we only needed to evaluate two configurations, we were able to allocate a more generous time budget of 10 hours.

In total, this set of 5 experiments would take $81 + 287 + 455 + 37 + 62 = 922$ days, if run in sequence. This would require more than two and a half years. Due to the substantial computation times required, we executed the experiments in parallel across 15 jobs on the same machine to optimize efficiency. Consequently, the first set of experiments took over 5 days, the second set needed just over 19 days, the third set extended beyond 30 days, the fourth set required more than 3 days, and the final set concluded in just over 4 days. This parallel processing approach significantly condensed the overall computation time, enabling us to conduct these exhaustive experiments within a manageable timeframe. In total, the

experiments took about 62 days when run in parallel. It was not possible to run more than 15 jobs at a time due to memory and TCP port constraints.

The experiments were conducted on a computer with the following specifications: Processor Intel(R) Xeon(R) Gold 6240R CPU @2.40GHz, 24 Core(s), 48 Logical Processor(s); RAM 192 GB; Operating System Windows 10 Pro for Workstations.

5.3 Statistical Analysis

Search-based algorithms inherently incorporate a degree of randomness. Consequently, adhering to the guidelines presented in Arcuri and Briand (2014) for executing SBSE experiments, we repeated each setting 10 times in the first set, 20 times in the fourth set, and 5 times in the fifth set. Regarding the third set of experiments, multiple runs were infeasible, since even a single run of the 729 configurations took more than 455 days. Performing multiple iterations helps mitigate the effects of randomness on the outcomes.

To interpret these outcomes, we conducted statistical analyses by utilizing the *Vargha-Delaney* effect size (\hat{A}_{12}) and the *Mann-Whitney-Wilcoxon U-test* (*p-value*) with a 95 % confidence level ($\alpha = 0.05$) (Arcuri and Briand 2014) to compare the results across different parameter configurations. The \hat{A}_{12} Vargha-Delaney value, which ranges from 0 to 1, measures the probability that one group outperforms another. A value of 0.5 indicates no difference between groups, while values above 0.5 show the better performance of the first group, with values closer to 1 indicating a stronger effect. Conversely, values less than 0.5 favor the second group, with values closer to 0 suggesting a stronger effect. The *p-value* represents the likelihood that the observed data would occur by chance if the null hypothesis were true. A *p-value* below α (i.e., 0.05) usually indicates statistically significant results, which means there is strong evidence to oppose the null hypothesis and the resulting values are less likely to be due to chance (Arcuri and Briand 2014). Note that, in this study, no multi-test corrections on the *p-values*, like the Bonferroni's one, has been used, because they are considered harmful (Arcuri and Briand 2014; Perneger 1998; Nakagawa 2004).

5.4 Results of the First Set of Experiments

5.4.1 Results of Re-tuning the Strategy of Enabling Feedback-Directed Sampling

The original study (Arcuri 2018) was published in 2018, and examined the performance of three configurations of FDS, i.e., $FDS \in \{\text{NONE}, \text{FOCUSED}, \text{LAST}\}$. It demonstrated that, in two cases (i.e., *proxyprint* and *rest-news*), FOCUSED provided statistically better outcomes. Yet, in two distinct situations (*rest-scs* and *scout-api*), FDS yielded statistically lower coverage (i.e., NONE was preferable). The study was uncertain regarding which value was the best, with LAST chosen as default based on the experiments on synthetic, manufactured examples.

Table 5 presents results of the replication (Arcuri 2018) for `feedbackDirectedSampling` parameter and Table 6 shows their pair comparison results. Based on *Average* and *Median* results (see *Average (original)* and *Median (original)* in Table 6) of the original case studies, NONE (i.e., not applying FDS) exhibited slightly better in terms of line cover-

Table 5 Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of enabling `feedbackDirectedSampling`

SUT	Line Cov%			Branch Cov%			Faults#		
	A	B	C	A	B	C	A	B	C
Original SUTs									
catwatch	45	45	44	18	18	18	27	26	26
features-service	82	82	82	26	25	25	33	33	33
proxyprint	53	53	52	13	13	12	83	81	81
rest-ncs	91	93	93	65	67	67	6	6	6
rest-news	66	66	66	23	24	24	8	8	8
rest-scs	80	86	86	46	51	51	9	12	12
scout-api	53	53	53	23	23	22	90	91	93
Average (original)	67	68	68	31	31	31	37	37	37
Avg Std Dev (original)	3	1	1	3	1	1	3	2	2
Median (original)	66	66	66	23	24	24	27	26	26
95% CI-LB (original)	66	68	68	30	31	31	36	36	36
95% CI-UB (original)	68	69	69	32	32	32	37	37	38
Other SUTs									
cwa-verification	47	47	47	9	9	9	8	8	8
genome-nexus	36	36	35	19	19	18	21	18	18
gestaohospital-rest	40	40	40	3	3	3	22	22	22
ind0	14	17	16	2	4	3	43	49	50
languagetool	36	37	38	14	14	15	6	6	7
market	48	48	47	3	3	3	19	19	20
ocvn-rest	36	35	35	6	6	6	542	551	545
restcountries	74	74	73	57	57	57	2	2	2
Average (other)	41	42	41	14	14	14	83	84	84
Avg Std Dev (other)	5	2	5	2	1	2	3	2	2
Median (other)	38	38	39	8	8	8	20	19	19
95% CI-LB (other)	41	42	41	14	14	14	82	84	83
95% CI-UB (other)	44	43	45	16	15	16	84	85	85
Average (all)	53	54	54	22	22	22	61	62	62
Avg Std Dev (all)	4	2	3	2	1	2	3	2	2
Median (all)	48	48	47	18	18	18	21	19	20
95% CI-LB (all)	53	54	54	21	22	22	60	61	61
95% CI-UB (all)	56	55	56	23	23	23	62	63	63

age, FOCUSED had a minor advantage for branch coverage, and LAST showed a minor advantage in fault detection. A comparison of the results from the eight additional case studies (see *Average (other)* and *Median (other)*) showed that, while there is no significant difference between the three configurations, FOCUSED had a slight overall advantage. It achieved slightly better branch coverage and almost the same line coverage and found faults compared to LAST and NONE. In a total of 15 REST APIs, there is no clear winner among the three settings in terms of all of the three metrics, reaffirming the same conclusions as in the original study (Golmohammadi et al. 2023).

Table 6 Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of enabling `feedbackDirectedSampling`

SUT	Line Cov			Branch Cov			Faults		
	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}
Original SUTs									
catwatch	0.49	0.58	0.59	0.51	0.62	0.60	0.55	0.56	0.51
features-service	0.48	0.48	0.49	0.56	0.53	0.47	0.71	0.65	0.48
proxyprint	0.48	0.56	0.58	0.56	0.57	0.53	0.56	0.58	0.54
rest-ncs	0.23	0.23	0.50	0.25	0.28	0.56	0.50	0.50	0.50
rest-news	0.40	0.40	0.49	0.45	0.45	0.49	0.56	0.54	0.48
rest-scs	0.23	<i>0.16</i>	<i>0.34</i>	<i>0.30</i>	<i>0.23</i>	0.38	0.23	0.23	0.50
scout-api	0.44	0.43	0.46	0.46	0.59	0.68	0.46	<i>0.33</i>	0.40
Average (original)	0.39	0.40	0.49	0.44	0.47	0.53	0.51	0.48	0.49
Median (original)	0.44	0.43	0.49	0.46	0.53	0.53	0.55	0.54	0.50
Other SUTs									
cwa-verification	0.45	0.45	0.50	0.41	0.44	0.53	0.43	0.44	0.52
genome-nexus	0.63	0.65	0.54	0.57	0.68	0.60	0.94	0.93	0.46
gestaohospital-rest	0.61	0.63	0.53	0.54	0.50	0.46	0.50	0.50	0.50
ind0	<i>0.26</i>	<i>0.32</i>	0.53	<i>0.27</i>	<i>0.33</i>	0.53	0.26	<i>0.25</i>	0.46
languagetool	0.46	0.36	<i>0.32</i>	0.45	<i>0.34</i>	<i>0.35</i>	0.49	0.37	0.37
market	0.48	0.59	0.60	0.47	0.60	0.61	0.52	0.46	0.44
ocvn-rest	0.57	0.51	0.44	0.60	0.54	0.44	<i>0.19</i>	0.40	0.71
restcountries	0.43	0.44	0.53	0.44	0.48	0.55	0.50	0.50	0.50
Average (other)	0.49	0.49	0.50	0.47	0.49	0.51	0.48	0.48	0.49
Median (other)	0.47	0.48	0.53	0.46	0.49	0.53	0.50	0.45	0.48
Average (all)	0.44	0.45	0.50	0.46	0.48	0.52	0.49	0.48	0.49
Median (all)	0.46	0.45	0.50	0.46	0.50	0.53	0.50	0.50	0.50

For instance, when comparing x with y , \hat{A}_{xy} , values in **bold** mean x is statistically significantly better than y , whereas values in *italic* mean y is statistically significantly better than x

5.4.2 Results of Re-tuning the Maximum Number of Rows Generated per SQL SELECT

The original study conducted in 2020 (Arcuri and Galeotti 2020) identified the optimal value for the parameter `maxSQLInitActionsPerMissingData` to be 5. This value was determined based on its ability to yield the highest average coverage across various test scenarios. Its performance in two out of five case studies, namely *catwatch* and *proxyprint*, excelled significantly.

Table 7 displays the outcomes of our replicated study, while Table 8 presents the comparative analysis of these results in pairs. According to Table 8, we found that a configuration of 5 for `maxSQLInitActionsPerMissingData` is indeed the most effective. This held true not only in terms of line coverage, which was the primary metric in the original study, but also when evaluating two additional metrics: branch coverage and the number of faults discovered. Our replication included the same five case studies as the original research.

However, according to the pair comparison results in Table 8, in our study, the 5 configuration did not achieve the highest coverage for the *proxyprint* case study, contrary to

Table 7 Line Coverage, Branch Coverage, and Faults achieved by different settings of the maximum number of rows generated per SQL SELECT (`maxSQLInitActionsPerMissingData`)

maxSqlInitActionsPerMissingData A=5, B=1, C=3									
SUT	Line Cov%			Branch Cov%			Faults#		
	A	B	C	A	B	C	A	B	C
Original SUTs									
catwatch	45	45	45	18	18	18	27	26	26
features-service	82	81	81	26	25	25	33	33	33
proxyprint	53	52	53	13	12	13	83	83	83
rest-news	66	66	65	23	24	23	8	8	8
scout-api	53	53	53	23	23	23	90	90	89
Average (original)	60	59	59	21	20	20	48	48	48
Avg Std Dev (original)	2	2	2	2	2	2	3	3	3
Median (original)	53	53	53	23	23	23	33	33	33
95% CI-LB (original)	59	59	59	20	20	20	47	47	47
95% CI-UB (original)	61	61	61	21	21	21	49	49	49
Other SUTs									
cwa-verification	47	47	47	9	9	9	8	8	8
genome-nexus	36	36	36	19	19	19	21	21	20
gestaohospital-rest	40	40	40	3	3	3	22	22	22
ind0	14	14	15	2	2	3	43	41	43
languagetool	36	36	36	14	14	14	6	7	6
market	48	47	47	3	3	3	19	19	19
ocvn-rest	36	35	36	6	6	6	542	540	541
rest-ncs	91	92	92	65	66	65	6	6	6
rest-scs	80	78	78	46	44	44	9	8	8
restcountries	74	73	73	57	56	56	2	2	2
Average (other)	50	50	50	22	22	22	68	67	68
Avg Std Dev (other)	5	4	3	3	2	2	2	3	3
Median (other)	43	43	43	11	12	12	14	14	14
95% CI-LB (other)	49	49	49	22	22	22	67	66	66
95% CI-UB (other)	53	52	52	24	24	23	69	69	69
Average (all)	53	53	53	22	22	22	61	61	61
Avg Std Dev (all)	4	3	3	2	2	2	3	3	3
Median (all)	48	47	47	18	18	18	21	21	20
95% CI-LB (all)	53	53	53	21	21	21	60	60	60
95% CI-UB (all)	56	55	55	23	23	23	62	62	62

what was reported in the original study. This discrepancy underscores the complexity and variability inherent in software testing environments.

Moreover, the robustness of the 5 configuration was further validated through additional case studies included in our replicated research. Across all these new scenarios, as shown in Table 8, the 5 setting yielded the highest results on average across all three metrics: line coverage, branch coverage, and fault detection. This consistency strengthens the argument for its efficacy and suggests that the findings of the original study are not only reproducible, but also applicable to a wider range of cases than originally tested.

Table 8 Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of the maximum number of rows generated per SQL SELECT (`maxSQLInitActionsPerMissingData`)
`maxSQLInitActionsPerMissingData A=5, B=1, C=3`

SUT	Line Cov			Branch Cov			Faults		
	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}
Original SUTs									
catwatch	0.50	0.53	0.53	0.57	0.56	0.51	0.61	0.59	0.48
features-service	0.56	0.56	0.51	0.59	0.58	0.50	0.54	0.53	0.49
proxyprint	0.55	0.46	0.41	0.59	0.46	0.38	0.48	0.49	0.50
rest-news	0.50	0.60	0.60	0.46	0.57	0.60	0.54	0.61	0.57
scout-api	0.46	0.47	0.52	0.41	0.47	0.57	0.50	0.56	0.56
Average (original)	0.52	0.52	0.51	0.52	0.53	0.51	0.53	0.55	0.52
Median (original)	0.50	0.53	0.52	0.57	0.56	0.51	0.54	0.56	0.50
Other SUTs									
cwa-verification	0.46	0.49	0.52	0.46	0.50	0.54	0.51	0.46	0.44
genome-nexus	0.57	0.50	0.42	0.51	0.51	0.47	0.46	0.57	0.60
gestaohospital-rest	0.65	0.61	0.45	0.52	0.46	0.44	0.52	0.50	0.48
ind0	0.46	0.40	0.44	0.52	0.43	0.42	0.54	0.46	0.43
languagetool	0.44	0.50	0.59	0.40	0.45	0.58	0.39	0.47	0.58
market	0.51	0.62	0.60	0.50	0.62	0.61	0.53	0.48	0.45
ocvn-rest	0.54	0.54	0.50	0.55	0.54	0.49	0.54	0.54	0.50
rest-ncs	0.40	0.43	0.53	0.40	0.44	0.55	0.50	0.50	0.50
rest-scs	0.53	0.61	0.56	0.55	0.59	0.51	0.57	0.55	0.46
restcountries	0.57	0.58	0.51	0.57	0.55	0.49	0.50	0.50	0.50
Average (other)	0.51	0.53	0.51	0.50	0.51	0.51	0.51	0.50	0.49
Median (other)	0.52	0.52	0.52	0.52	0.50	0.50	0.52	0.50	0.49
Average (all)	0.51	0.53	0.51	0.51	0.52	0.51	0.52	0.52	0.50
Median (all)	0.51	0.53	0.52	0.52	0.51	0.51	0.52	0.50	0.50

For instance, when comparing x with y , \hat{A}_{xy} , values in **bold** mean x is statistically significantly better than y , whereas values in *italic* mean y is statistically significantly better than x

5.4.3 Results of Re-tuning the Probability of Applying SQL Handling to Prepare Resources

In the original study (Zhang and Arcuri 2021a), a 0.5 setting for `probOfApplySQLActionToCreateResources` was found to be the most effective in three specific projects: *rest-news*, *proxyprint*, and *scout-api*. This implies that relying solely on HTTP methods such as POST or PUT for adding new resources might not be ideal in these instances. Consequently, a greater likelihood of employing SQL commands directly appeared to yield superior results. Conversely, for projects like *features-service* and *catwatch*, a diminished setting proved to be more advantageous (i.e., 0.1 and 0.3, respectively). Taking these findings into account, the developers of EvoMASTER set 0.5 as the standard value for this parameter.

Our study replicated the original to assess the consistency of results and establish stronger conclusions. Table 9 presents the results from our replication study, and Table 10 provides a pairwise comparison of these outcomes. As demonstrated in Table 10, the outcomes regarding optimal code coverage are varied. Although a 0.5 setting was effective for *rest-news*, it did not yield the best results for *proxyprint* and *scout-api*. In these cases, 0.3 and 0.1 were the respective optimal settings. This represents a deviation from the original

Table 9 Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of applying SQL handling to prepare resources (`probOfApplySQLActionToCreateResources`)

probOfApplySQLActionToCreateResources A=0.5, B=0.1, C=0.3									
SUT	Line Cov%			Branch Cov%			Faults#		
	A	B	C	A	B	C	A	B	C
Original SUTs									
catwatch	45	45	44	18	18	17	27	27	26
features-service	82	81	81	26	25	25	33	34	33
proxyprint	53	53	54	13	13	14	83	83	84
rest-news	66	65	65	23	22	23	8	8	8
scout-api	53	53	53	23	23	23	90	92	90
Average (original)	60	59	59	21	20	20	48	49	48
Avg Std Dev (original)	2	2	2	2	2	2	3	3	2
Median (original)	53	53	54	23	22	23	33	34	33
95% CI-LB (original)	59	59	59	20	20	20	47	48	47
95% CI-UB (original)	61	61	61	21	21	21	49	50	49
Other SUTs									
cwa-verification	47	47	47	9	9	9	8	8	8
genome-nexus	36	36	36	19	20	19	21	21	21
gestaohospital-rest	40	40	40	3	3	3	22	22	22
ind0	14	14	15	2	2	3	43	42	42
languagetool	36	38	36	14	15	14	6	6	7
market	48	47	47	3	3	3	19	19	20
ocvn-rest	36	36	35	6	6	6	542	543	541
rest-ncs	91	92	92	65	65	65	6	6	6
rest-scs	80	78	78	46	43	43	9	8	8
restcountries	74	73	73	57	56	56	2	2	2
Average (other)	50	50	50	22	22	22	68	68	68
Avg Std Dev (other)	5	3	6	3	2	3	2	3	3
Median (other)	43	43	43	11	12	12	14	14	14
95% CI-LB (other)	49	50	49	22	22	22	67	67	66
95% CI-UB (other)	53	52	54	24	23	24	69	69	69
Average (all)	53	53	53	22	22	22	61	61	61
Avg Std Dev (all)	4	3	5	2	2	3	3	3	3
Median (all)	48	47	47	18	18	17	21	21	21
95% CI-LB (all)	53	53	52	21	21	21	60	60	60
95% CI-UB (all)	56	55	56	23	23	23	62	62	62

study, where 0.5 was universally the most effective. In our replication, *feature-service* and *catwatch* showed improved code coverage with a 0.5 setting, unlike in the original study.

Overall, considering the five SUTs from the initial study, a 0.5 setting was generally most effective in maximizing code coverage. Additionally, our research discovered an unexpected result in fault detection. The 0.1 setting led to the discovery of more faults, warranting further analysis. By extending our research to include 10 more SUTs, we observed that 0.5 slightly improved line and branch coverage, while fault detection remained constant (Golmohammadi et al. 2023).

Table 10 Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of applying SQL handling to prepare resources (`probOfApplySQLActionToCreateResources`)

<code>probOfApplySQLActionToCreateResources A=0.5, B=0.1, C=0.3</code>									
SUT	Line Cov			Branch Cov			Faults		
	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}
Original SUTs									
catwatch	0.53	0.65	0.60	0.53	0.69	0.64	0.40	0.64	0.73
features-service	0.54	0.53	0.47	0.54	0.59	0.54	0.39	0.52	0.63
proxyprint	0.46	0.38	0.45	0.47	0.45	0.49	0.47	0.44	0.46
rest-news	0.70	0.60	0.39	0.77	0.63	0.35	0.62	0.44	0.33
scout-api	0.48	0.54	0.54	0.43	0.48	0.54	0.43	0.49	0.57
Average (original)	0.54	0.54	0.49	0.55	0.57	0.51	0.46	0.50	0.54
Median (original)	0.53	0.54	0.47	0.53	0.59	0.54	0.43	0.49	0.57
Other SUTs									
cwa-verification	0.47	0.47	0.50	0.45	0.51	0.56	0.47	0.41	0.44
genome-nexus	0.51	0.54	0.54	0.46	0.50	0.57	0.53	0.50	0.48
gestaohospital-rest	0.62	0.62	0.51	0.51	0.50	0.50	0.50	0.50	0.50
ind0	0.45	0.40	0.42	0.50	0.43	0.42	0.55	0.53	0.48
languagetool	0.38	0.46	0.59	0.36	0.44	0.58	0.44	0.40	0.46
market	0.56	0.59	0.51	0.56	0.59	0.51	0.54	0.44	0.41
ocvn-rest	0.42	0.53	0.61	0.46	0.56	0.58	0.48	0.54	0.55
rest-ncs	0.45	0.43	0.46	0.46	0.46	0.48	0.50	0.50	0.50
rest-scs	0.57	0.57	0.50	0.58	0.57	0.51	0.58	0.58	0.51
restcountries	0.53	0.60	0.55	0.53	0.59	0.55	0.50	0.50	0.50
Average (other)	0.50	0.52	0.52	0.49	0.51	0.53	0.51	0.49	0.48
Median (other)	0.49	0.54	0.51	0.48	0.51	0.53	0.50	0.50	0.49
Average (all)	0.51	0.53	0.51	0.51	0.53	0.52	0.49	0.49	0.50
Median (all)	0.51	0.54	0.51	0.50	0.51	0.54	0.50	0.50	0.50

For instance, when comparing x with y , \hat{A}_{xy} , values in **bold** mean x is statistically significantly better than y , whereas values in *italic* mean y is statistically significantly better than x

Evaluating the collective data from all 15 REST APIs, we determined that a 0.5 setting consistently provided the best overall performance, aligning with the conclusions of the original study (Golmohammadi et al. 2023).

5.4.4 Results of Re-tuning the Probability of Applying Random Sampling

In the initial research (Zhang and Arcuri 2021b), the focus was on two key parameters: `probOfRandomSampling` and `focusedSearchActivationTime`. This study aims to expand and replicate the previous work by adding eight more case studies and modifying these two parameters. In the current study, we treated these parameters as separate variables and evaluated each independently.

The original research suggested a 0.5 setting as the most effective for both parameters. Yet, our broader study uncovered slightly different results. For the `probOfRandomSampling` parameter, examining the original SUTs presented in Table 12 reveals that a 0.8 setting produced a relatively higher branch coverage compared to the other two settings, including the default setting (i.e., 0.5), with a statistically significant higher branch cover-

age in *rest-news*. However, it had slightly lower average and median values in their pairwise comparison (i.e., 0.48 and 0.47, respectively) for the default setting. In terms of line coverage and faults, 0.8 performed more or less the same.

Across the other SUTs, there is still no clear winner. The default value appears to be the better choice for line coverage. Even though, according to Table 12, the average and median pairwise comparison of 0.8 and 0.5 did not indicate a clear advantage for either, 0.5 showed a statistically significant higher number (i.e., 0.66, which is written in bold in Table 12), making the default setting the better option.

On the contrary, 0.8 appears to be a slightly better choice in terms of faults across other SUTs, showing significantly higher performance in *ocvn-rest*. In terms of branch coverage, the results are somewhat surprising, as both the default value and 0.8 were marginally outperformed by 0.2, which showed statistically significant better performance in at least one SUT, despite the almost equal average and median values.

In the end, there is no clear winner among these compared configurations. However, 0.8 appears to be slightly better.

5.4.5 Results of Re-tuning the Time of Activating Focused Search

As we evaluate the `focusedSearchActivationTime` parameter, we observe that while a setting of 0.5 appeared highly effective for certain SUTs, like *catwatch* and *feature-service*, a wider view encompassing other SUTs and the quantity of detected faults indicates a contrasting narrative.

Table 13 shows the actual results from our replication study, and Table 14 presents a pairwise comparison of the results. As presented in Table 14, in contrast to the conclusions of the original research (Zhang and Arcuri 2021b), a 0.5 value did not excel beyond others. This trend was primarily observed across the original case studies, where 0.8 outperformed the default value on average, with statistically significant improvements in code coverage for two SUTs (i.e., *proxyprint* and *scout-api*). In the original SUTs, both values performed similarly in terms of fault detection, with no statistically significant differences across the SUTs. When considering the supplementary SUTs, 0.8 and the default value showed comparable performance in terms of code coverage. However, 0.8 was more effective in fault detection, as shown in Table 14. Overall, 0.8 appears to be the most effective value compared to both the default and the other tested values.

5.4.6 Results of Re-tuning the Probability of Base Taint Analysis

The original study was not quite decisive on determining the optimal setting for the `baseTaintAnalysisProbability` (Arcuri and Galeotti 2021). On average, 0.9 performed better than other values in terms of line coverage, although this was not the case for all the case studies. For instance, the utilization of input tracking system deteriorated the line coverage for *features-service*. Moreover, for *proxyprint*, the input tracking system seemed to have some side effects as the lowest value (i.e., 0.1) yielded the highest coverage and the higher the value of the parameter was, the lower line coverage was achieved. On the other hand, the lower values resulted in detecting more faults: the higher the parameter value was, the fewer faults were found. However, the difference in the number of found faults was insignificant and the configuration of 0.9 was suggested to be more reasonable.

The results of the current study is shown in Table 15. According to Table 16 which presents the pair comparison of the different settings, the suggested value by the original study (i.e., 0 . 9) resulted in higher line and branch coverage, for both original and additional SUTs on average. In addition, the setting of 0 . 5 performed better in terms of code coverage than that of 0 . 1. This means that the trend in which the higher the probability of base taint analysis is, the higher code coverage is achieved, is being witnessed in our study too. Moreover, similar to the original study, the setting of 0 . 9 did not result in detecting higher number of faults either. This observation is somewhat similar across both original and additional case studies.

Upon consideration of all factors, the overall results are similar to the findings of the original study as the previously suggested value (i.e., 0 . 9) led to higher code coverage and slightly fewer faults.

5.4.7 Findings Summary for RQ1

In the case of `feedbackDirectedSampling` from the first replicated study, a slight advantage of FOCUSED in terms of branch coverage was witnessed. However, by taking all the three metrics into account, none of the settings were a clear winner, which reaffirmed the outcomes of the original study. Regarding the `probOfApplySQLActionToCreateResources`, a similar pattern to the original study was discovered as a setting of 0 . 5 yielded higher line and branch coverage. Nevertheless, it did not lead to better fault detection as the number of found faults were very similar, as shown in Table 10. On the other hand, the results differed for `probOfRandomSampling` and `focusedSearchActivationTime`. Contrary to the original articles that found 0 . 5 to be the optimum, here we discovered that 0 . 8 performed better across multiple case studies (Golmohammadi et al. 2023). For `probOfRandomSampling`, the difference was marginal, as both 0 . 8 and the default value produced almost similar results, except for a slightly better performance of 0 . 8 in branch coverage across the original SUTs and in fault detection across the supplementary SUTs.

For `focusedSearchActivationTime`, the difference was more notable. Across the original studies, 0 . 8 performed better in both line and branch coverage while achieving almost the same fault detection performance. In contrast, across the other SUTs, 0 . 8 produced results nearly identical to the default but showed better fault detection, making it a more reasonable option overall.

The results of our replication for the two parameters which were not present in Golmohammadi et al. (2023) (i.e., `maxSQLInitActionsPerMissingData` and `baseTaintAnalysisProbability`) were consistent with the original studies' findings (Arcuri and Galeotti 2020, 2021).

The narrow confidence intervals presented in the tables showing the actual values, Tables 5, 7, 9, 11, 13, and 15, including the 95% CI-LB (Confidence Interval lower bound) and 95% CI-UB (Confidence Interval upper bound), along with low standard deviation values, suggest low variability in the data. The higher mean values relative to the median imply right-skewed data, where a few high values pull the average up. This is most notable in fault detection (the number of faults discovered), primarily caused by `ocvn-rest`, which has a notably higher number of endpoints compared to other SUTs, as shown in Table 2. This reflects a stable yet slightly asymmetric distribution.

Table 11 Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of applying random sampling (`probOfRandomSampling`)`probOfRandomSampling A=0.5, B=0.2, C=0.8`

SUT	Line Cov%			Branch Cov%			Faults#		
	A	B	C	A	B	C	A	B	C
Original SUTs									
rest-ncs	91	91	90	65	64	64	6	6	6
rest-scs	80	78	78	46	44	46	9	8	9
rest-news	66	65	66	23	23	24	8	8	8
catwatch	45	43	46	18	17	19	27	26	27
features-service	82	81	81	26	25	24	33	34	34
proxyprint	53	51	54	13	12	13	83	83	85
scout-api	53	53	53	23	23	23	90	89	90
Average (original)	67	66	67	31	30	30	37	36	37
Avg Std Dev (original)	3	3	4	3	3	3	3	2	2
Median (original)	66	65	66	23	23	24	27	26	27
95% CI-LB (original)	66	65	66	30	29	29	36	35	36
95% CI-UB (original)	68	67	68	32	31	31	37	37	38
Other SUTs									
cwa-verification	47	47	47	9	9	9	8	8	8
genome-nexus	36	36	36	19	19	19	21	21	21
gestaohospital-rest	40	40	40	3	3	3	22	22	22
ind0	14	16	13	2	3	2	43	44	41
languagetool	36	36	37	14	14	14	6	5	6
market	48	48	47	3	3	3	19	19	19
ocvn-rest	36	36	36	6	6	6	542	509	569
restcountries	74	73	73	57	56	57	2	2	2
Average (other)	41	41	41	14	14	14	83	79	86
Avg Std Dev (other)	5	5	3	2	2	1	3	4	3
Median (other)	38	38	38	8	8	8	20	20	20
95% CI-LB (other)	41	41	41	14	14	14	82	77	85
95% CI-UB (other)	44	45	43	16	16	15	84	80	87
Average (all)	53	53	53	22	21	22	61	59	63
Avg Std Dev (all)	4	4	3	2	2	2	3	3	3
Median (all)	48	48	47	18	17	19	21	21	21
95% CI-LB (all)	53	52	52	21	21	21	60	58	62
95% CI-UB (all)	56	56	55	23	23	23	62	60	64

RQ1: Although 4 out of the 6 initial parameter recommendations are still applicable, the outcomes for two specific parameters suggest that they should be reevaluated as the development and evolution of the EVOMASTER tool progresses.

5.4.8 Findings Summary for RQ2

A consistent pattern was observed when analyzing the results of the two additional replicated studies (Arcuri and Galeotti 2020, 2021) as well as the four ones from Golmohammadi et al. (2023). It can be seen that the results derived from the original case studies were generally affirmed by the other SUTs (i.e., the newly added ones). For example, even though

Table 12 Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of applying random sampling (`probOfRandomSampling`)
probOfRandomSampling A=0.5, B=0.2, C=0.8

SUT	Line Cov			Branch Cov			Faults		
	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}
Original SUTs									
rest-ncs	0.52	0.48	0.48	0.52	0.51	0.50	0.50	0.55	0.55
rest-scs	0.60	0.50	0.43	0.63	0.47	0.37	0.60	0.47	0.38
rest-news	0.56	0.52	0.43	0.71	<i>0.33</i>	<i>0.17</i>	0.58	0.45	0.38
catwatch	0.75	0.40	<i>0.16</i>	0.80	0.41	<i>0.11</i>	0.67	0.44	0.27
features-service	0.68	0.57	<i>0.35</i>	0.57	0.64	0.58	0.51	0.45	0.45
proxyprint	0.64	0.38	<i>0.23</i>	0.60	0.45	0.36	0.49	0.37	0.35
scout-api	0.56	0.56	0.50	0.51	0.56	0.55	0.57	0.49	0.38
Average (original)	0.62	0.49	0.37	0.62	0.48	0.38	0.56	0.46	0.39
Median (original)	0.60	0.50	0.43	0.60	0.47	0.37	0.57	0.45	0.38
Other SUTs									
cwa-verification	0.43	0.44	0.52	0.47	0.45	0.50	0.47	0.46	0.49
genome-nexus	0.52	0.48	0.44	0.55	0.44	0.40	0.49	0.49	0.51
gestaohospital-rest	0.62	0.66	0.54	0.52	0.52	0.50	0.52	0.50	0.48
ind0	<i>0.31</i>	0.50	0.70	<i>0.35</i>	<i>0.59</i>	0.72	0.49	0.57	0.55
languagetool	0.47	0.41	0.45	0.44	0.41	0.47	0.52	0.50	0.48
market	0.46	0.52	0.56	0.47	0.52	0.56	0.49	0.48	0.49
ocvn-rest	0.54	0.47	0.45	0.53	0.50	0.48	0.98	<i>0.00</i>	<i>0.00</i>
restcountries	0.57	0.55	0.48	0.56	0.55	0.49	0.50	0.50	0.50
Average (other)	0.49	0.51	0.52	0.49	0.50	0.51	0.56	0.44	0.44
Median (other)	0.50	0.49	0.50	0.50	0.51	0.49	0.50	0.50	0.49
Average (all)	0.55	0.50	0.45	0.55	0.49	0.45	0.56	0.45	0.42
Median (all)	0.56	0.50	0.45	0.53	0.50	0.49	0.51	0.48	0.48

For instance, when comparing x with y , \hat{A}_{xy} , values in **bold** mean x is statistically significantly better than y , whereas values in *italic* mean y is statistically significantly better than x

there was an anomaly in the replication result for `probOfApplySQLActionToCreateResources`, where the configuration of 0.1 resulted in marginally more faults for the original SUTs, this difference became neutralized when considering the results from other SUTs.

RQ2: Incorporating more SUTs into our replication research largely validated the outcomes observed with the SUTs in the original studies. This strengthens the external validity of the findings.

5.5 Results of the Second Set of Experiments

This section presents the results of the second set of experiments, which aimed to assess whether expanding the parameter space influences the validity of previous findings. By testing a finer granularity of values, we sought to determine if the previously identified optimal settings remained consistent or if better configurations emerged. While some values were already tested in the first set, we include them here again for comparison. However, the focus of this section is on the newly introduced configurations.

Table 13 Line Coverage, Branch Coverage, and Faults achieved by different settings of the time of activating focused search (`focusedSearchActivationTime`)

SUT	Line Cov%			Branch Cov%			Faults#		
	A	B	C	A	B	C	A	B	C
Original SUTs									
rest-ncs	91	91	91	65	64	65	6	6	6
rest-scs	80	79	78	46	45	44	9	9	9
rest-news	66	65	65	23	23	23	8	7	8
catwatch	45	44	46	18	17	18	27	25	27
features-service	82	81	82	26	25	26	33	33	34
proxyprint	53	51	55	13	12	14	83	84	85
scout-api	53	52	54	23	23	24	90	87	92
Average (original)	67	66	67	31	30	31	37	36	37
Avg Std Dev (original)	3	3	3	3	3	2	3	2	2
Median (original)	66	65	65	23	23	24	27	25	27
95% CI-LB (original)	66	65	66	30	29	30	36	35	36
95% CI-UB (original)	68	68	69	32	31	32	37	37	38
Other SUTs									
cwa-verification	47	47	47	9	9	9	8	8	8
genome-nexus	36	35	36	19	18	20	21	20	20
gestaohospital-rest	40	40	40	3	3	3	22	22	22
ind0	14	13	15	2	2	3	43	39	43
languagetool	36	35	36	14	14	14	6	7	7
market	48	47	47	3	3	3	19	19	20
ocvn-rest	36	35	35	6	6	6	542	518	561
restcountries	74	73	73	57	56	56	2	2	2
Average (other)	41	41	41	14	14	14	83	79	85
Avg Std Dev (other)	5	6	2	2	3	1	3	3	3
Median (other)	38	37	38	8	8	8	20	19	20
95% CI-LB (other)	41	40	41	14	14	14	82	78	84
95% CI-UB (other)	44	45	43	16	16	15	84	81	87
Average (all)	53	53	53	22	21	22	61	59	63
Avg Std Dev (all)	4	5	3	2	3	2	3	3	3
Median (all)	48	47	47	18	17	18	21	20	20
95% CI-LB (all)	53	52	53	21	21	22	60	58	62
95% CI-UB (all)	56	56	55	23	23	23	62	60	64

In this set of experiments, we evaluated five parameters: `maxSQLInitActionsPerMissingData`, `probOfApplySQLActionToCreateResources`, `probOfRandomSampling`, `focusedSearchActivationTime`, and `baseTaintAnalysisProbability`. Unlike this first set, `feedbackDirectedSampling` was not considered in this set as it is of type `Enum` and there were no additional values to be evaluated.

The first parameter, an integer, was tested with three additional values, increasing its range to five values plus the default. The remaining four parameters, which are of type double and represent probabilities, were tested across 11 values, covering the entire range from 0 to 1 in increments of 0.1. The experiment involved evaluating 46 different parameter

Table 14 Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of the time of activating focused search (focusedSearchActivationTime)

SUT	Line Cov			Branch Cov			Faults		
	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}
Original SUTs									
rest-ncs	0.49	0.48	0.50	0.51	0.47	0.47	0.52	0.50	0.48
rest-scs	0.54	0.60	0.56	0.57	0.59	0.53	0.53	0.55	0.52
rest-news	0.62	0.55	0.42	0.52	0.49	0.47	0.67	0.48	0.32
catwatch	0.67	0.42	0.29	0.74	0.45	0.20	0.73	0.38	0.16
features-service	0.60	0.47	0.35	0.57	0.44	0.36	0.70	0.49	0.30
proxyprint	0.66	0.29	<i>0.17</i>	0.67	0.35	0.20	0.44	0.40	0.44
scout-api	0.57	<i>0.34</i>	<i>0.25</i>	0.50	<i>0.29</i>	<i>0.26</i>	0.65	0.36	0.17
Average (original)	0.59	0.45	0.36	0.58	0.44	0.36	0.61	0.45	0.34
Median (original)	0.60	0.47	0.35	0.57	0.45	0.36	0.65	0.48	0.32
Other SUTs									
cwa-verification	0.48	0.41	0.43	0.49	0.43	0.45	0.55	0.44	0.39
genome-nexus	0.71	0.51	<i>0.27</i>	0.67	0.50	0.28	0.58	0.53	0.44
gestaohospital-rest	0.72	0.59	0.37	0.52	0.52	0.50	0.50	0.50	0.50
ind0	0.57	0.38	<i>0.32</i>	0.64	0.43	<i>0.30</i>	0.65	0.52	0.37
languagetool	0.51	0.47	0.44	0.48	0.46	0.48	0.49	0.39	0.40
market	0.55	0.55	0.51	0.54	0.55	0.51	0.65	0.39	0.26
ocvn-rest	0.60	0.45	0.36	0.60	0.56	0.46	0.99	<i>0.03</i>	<i>0.00</i>
restcountries	0.62	0.61	0.47	0.60	0.58	0.48	0.50	0.50	0.50
Average (other)	0.59	0.50	0.40	0.57	0.50	0.43	0.61	0.41	0.36
Median (other)	0.58	0.49	0.40	0.57	0.51	0.47	0.56	0.47	0.40
Average (all)	0.59	0.48	0.38	0.57	0.47	0.40	0.61	0.43	0.35
Median (all)	0.60	0.47	0.37	0.57	0.47	0.46	0.58	0.48	0.39

For instance, when comparing x with y , \hat{A}_{xy} , values in **bold** mean x is statistically significantly better than y , whereas values in *italic* mean y is statistically significantly better than x

settings across 15 REST APIs, each executed with a one-hour time budget and repeated 10 times, leading to a total computation time exceeding 287 days. We picked the best performing value from this set of experiments for each parameter and compared it with the best value from the previous set by looking at their Vargha-Delaney effect-sizes and p-values of U-tests. In case we did not find any better performing value to compare with the best value from the previous set, we just chose the second best performing value. The comparison results are presented in Tables 40, 41, and 42.

5.5.1 Results of Re-tuning the Maximum Number of Rows Generated per SQL SELECT with Expanded Values

The results of the second set of experiments, where we evaluated an expanded range of values for `maxSQLInitActionsPerMissingData`, reveal some deviations from previous findings. Unlike the original study (Arcuri and Galeotti 2020) and the first set of experiments in Section 5.4.2, which identified 5 as the best setting, our results indicate that 1 consistently performed better across multiple metrics.

Table 15 Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of base taint analysis (`baseTaintAnalysisProbability`)

SUT	Line Cov%			Branch Cov%			Faults#		
	A	B	C	A	B	C	A	B	C
Original SUTs									
catwatch	45	45	45	18	18	18	27	26	26
features-service	82	81	81	26	25	25	33	34	34
languagetool	36	34	36	14	13	14	6	7	7
proxyprint	53	53	52	13	13	12	83	85	85
rest-ncs	91	91	91	65	65	65	6	6	6
rest-news	66	65	66	23	23	23	8	8	8
restcountries	74	73	72	57	56	56	2	2	2
rest-scs	80	77	78	46	45	44	9	10	9
scout-api	53	52	53	23	23	23	90	89	89
Average (original)	64	64	64	32	31	31	29	30	29
Avg Std Dev (original)	5	6	3	3	3	3	2	2	2
Median (original)	66	65	66	23	23	23	9	10	9
95% CI-LB (original)	64	63	63	31	30	31	28	29	29
95% CI-UB (original)	68	67	66	33	33	32	30	30	30
Other SUTs									
cwa-verification	47	47	47	9	9	9	8	8	8
genome-nexus	36	35	36	19	18	19	21	22	21
gestaohospital-rest	40	40	40	3	3	3	22	22	22
ind0	14	10	14	2	1	2	43	41	43
market	48	47	48	3	3	3	19	20	20
ocvn-rest	36	35	35	6	6	6	542	545	545
Average (other)	37	36	37	7	7	7	109	110	110
Avg Std Dev (other)	2	2	2	1	1	1	3	3	4
Median (other)	38	38	38	5	5	5	21	22	21
95% CI-LB (other)	36	35	36	7	6	7	108	108	108
95% CI-UB (other)	37	37	37	8	7	7	110	111	111
Average (all)	53	52	53	22	21	22	61	62	62
Avg Std Dev (all)	4	4	3	2	2	2	3	3	3
Median (all)	48	47	48	18	18	18	21	22	21
95% CI-LB (all)	53	52	52	21	21	21	60	61	60
95% CI-UB (all)	56	55	54	23	23	22	62	63	63

For line coverage, according to Table 25, both 1 and 7 emerged as the top-performing values, achieving 54% and 53% on average, respectively. While 1 provided the highest overall average, 7 was superior in 10 out of 15 SUTs, compared to 9 for 1, suggesting that 7 could still be competitive in certain cases. For *branch coverage*, as shown in Table 26, 1 and 3 outperformed both the default value 5 and the extended values (i.e., 7, 15, 20), achieving 22% on average and being the best setting in 10 SUTs. However, 1 had a slightly higher median value, giving it a minor edge over 3. In terms of *fault detection*, as presented in Table 27, 1 was the best-performing setting, finding 61 faults on average and being the superior configuration in 9 SUTs. By comparing 1 and 5 across all three metrics in Tables 40, 41, and 42, it is evident that 1 performs better overall. Overall, while the differ-

Table 16 Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of the probability of base taint analysis (`baseTaintAnalysisProbability`)

baseTaintAnalysisProbability A=0.9, B=0.1, C=0.5									
SUT	Line Cov			Branch Cov			Faults		
	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}	\hat{A}_{AB}	\hat{A}_{AC}	\hat{A}_{BC}
Original SUTs									
catwatch	0.53	0.54	0.50	0.60	0.60	0.49	0.60	0.55	0.45
features-service	0.61	0.50	0.38	0.60	0.53	0.43	0.45	0.42	0.46
languagetool	0.55	0.47	0.41	0.54	0.44	0.41	0.46	0.44	0.49
proxyprint	0.43	0.52	0.61	0.53	0.55	0.56	0.39	0.39	0.49
rest-ncs	0.53	0.46	0.42	0.51	0.51	0.49	0.52	0.52	0.50
rest-news	0.60	0.54	0.44	0.54	0.52	0.47	0.54	0.47	0.43
restcountries	0.61	0.70	0.56	0.60	0.70	0.58	0.50	0.50	0.50
rest-scs	0.57	0.59	0.49	0.55	0.59	0.54	0.46	0.51	0.55
scout-api	0.61	0.58	0.52	0.49	0.52	0.54	0.54	0.56	0.52
Average (original)	0.56	0.54	0.48	0.55	0.55	0.50	0.50	0.48	0.49
Median (original)	0.57	0.54	0.49	0.54	0.53	0.49	0.50	0.50	0.49
Other SUTs									
cwa-verification	0.56	0.50	0.43	0.59	0.52	0.43	0.62	0.53	0.41
genome-nexus	0.65	0.53	0.38	0.65	0.50	0.34	0.25	0.43	0.69
gestaohospital-rest	0.48	0.61	0.63	0.52	0.52	0.50	0.50	0.50	0.50
ind0	0.77	0.46	0.20	0.87	0.60	0.17	0.60	0.45	0.36
market	0.52	0.49	0.49	0.53	0.49	0.47	0.45	0.46	0.50
ocvn-rest	0.70	0.57	0.38	0.59	0.59	0.51	0.40	0.42	0.51
Average (other)	0.61	0.53	0.42	0.62	0.54	0.40	0.47	0.46	0.49
Median (other)	0.60	0.51	0.40	0.59	0.52	0.45	0.47	0.45	0.50
Average (all)	0.58	0.54	0.46	0.58	0.55	0.46	0.49	0.48	0.49
Median (all)	0.57	0.53	0.44	0.55	0.52	0.49	0.50	0.47	0.50

For instance, when comparing x with y , \hat{A}_{xy} , values in **bold** mean x is statistically significantly better than y , whereas values in *italic* mean y is statistically significantly better than x

ences between these two values are noticeable, they are not statistically significant. Notably, none of the expanded values (7, 15, or 20) were consistently the best across all metrics, although 7 showed strong performance in line coverage.

5.5.2 Results of Re-tuning the Probability of Applying SQL Handling to Prepare Resources with Expanded Values

The second set of experiments provided new insights into the impact of tuning `probOfApplySQLActionToCreateResources`. While the original study (Zhang and Arcuri 2021a) and the first set of experiments in Section 5.4.3 both identified 0.5 as the best setting, our extended analysis suggests that 0.2 may be a more effective choice.

For line coverage, according to Table 28, 0.2 was the best-performing value, achieving an average coverage of 54%. Similarly, as displayed in Table 29, for branch coverage, 0.2 remained the highest on average at 22%, though 1 appeared slightly better, being the superior setting in three additional SUTs, particularly in the original SUTs. A similar trend was observed in fault detection, where 1 outperformed 0.2 in two more SUTs, as presented in Table 30.

Despite 1 showing a slight advantage in some cases, 0.2 appears to be the more rational choice. Setting this parameter which represents a probability to 1 might not be ideal. First of all, it eliminates diversity in test generation by always applying SQL-based resource creation, potentially missing alternative testing paths. It could also lead to unrealistic test scenarios if an API is primarily designed for resource creation through HTTP methods. Moreover, it may introduce unintended side effects by bypassing certain business logic checks enforced at the API level. Thus, while 0.2 does not completely dominate across all metrics, it offers a more balanced and practical trade-off between maximizing test effectiveness and maintaining realistic test scenarios. However, comparing 0.2 with 0.5 in Tables 40, 41, and 42 does not show any statistically significant advantage.

5.5.3 Results of Re-tuning the Probability of Applying Random Sampling with Expanded Values

In Section 5.4.4, we identified 0.8 as the best setting for `probOfRandomSampling`, which differed from the original study (Zhang and Arcuri 2021b) that found 0.5 to be optimal. In this second set, where we expanded the range of values from 0 to 1 in increments of 0.1, 0.8 continued to perform best in terms of code coverage.

For both line and branch coverage, as shown in Tables 31 and 32, 0.8 achieved the highest results, confirming its effectiveness in maximizing exploration while maintaining efficient search performance. In terms of fault detection, according to Table 33, 0.8 found slightly fewer faults compared to 1, but the difference was marginal.

Overall, despite evaluating a finer granularity of values, 0.8 remains the most effective setting. These findings reinforce the conclusion from the first set of experiments, suggesting that 0.8 provides the best balance between exploration and exploitation in the search process.

5.5.4 Results of Re-tuning the Time of Activating Focused Search with Expanded Values

The results for `focusedSearchActivationTime` followed the same pattern as `probOfRandomSampling`. In Section 5.4.5, we found 0.8 to be the best-performing value, differing from the original study that suggested 0.5 (Zhang and Arcuri 2021b). In this second set, where we tested values from 0 to 1 in increments of 0.1, 0.8 continued to yield the highest code coverage.

As displayed in Tables 34 and 35, for both line and branch coverage, 0.8 remained the best setting, demonstrating its effectiveness in optimizing the transition from exploration to exploitation. In terms of fault detection, Table 36 shows that 0.8 performed slightly worse than 1, but the difference was minimal.

Overall, the results confirm that 0.8 is the most effective setting for this parameter. Despite applying a finer granularity of values, 0.8 consistently outperformed other settings, which reinforces its advantage in balancing broad exploration with focused search.

5.5.5 Results of Re-tuning the Probability of Base Taint Analysis with Expanded Values

In both the original study (Arcuri and Galeotti 2021) and Section 5.4.6, 0.9 was identified as the best setting for `baseTaintAnalysisProbability`. In this second set, where we expanded the parameter values, the results indicate that no single value consistently outperforms across all metrics.

For line coverage, according to Table 37, 0.9 remained one of the top-performing values, along with all other settings except 0.0, which resulted in noticeably lower coverage. In branch coverage, as shown in Table 38, 0.3 performed better, achieving the highest average across the SUTs and being the superior setting in 10 SUTs. In terms of fault detection, as it is presented in Table 39, 0.4 yielded the highest number of detected faults, with an average of 61. Overall, the results show that there is no clear winner for this parameter, but some of the newly tested values performed better than the original default setting in specific metrics.

5.5.6 Findings Summary for RQ3

Expanding the parameter space revealed some new values that outperformed previously identified optimal settings in certain metrics. `maxSQLInitActionsPerMissingData` and `probOfApplySQLActionToCreateResources` showed better-performing alternatives, while `probOfRandomSampling` and `focusedSearchActivationTime` retained 0.8 as the best setting. `baseTaintAnalysisProbability` had no single dominant value, with different settings excelling in different metrics. However, while some new values appeared to perform better, none of the observed differences were statistically significant. The observed variation could be attributed to the inherent randomness in the search process, highlighting the need for further investigation into the stability of parameter tuning results.

RQ3: Expanding the parameter values did not lead to finding any outstanding values compared to the results from the first set. This strengthens the external validity of the findings.

5.6 Results of the Third Set of Experiments

In this phase of our study, we moved beyond the individual parameter analysis by extending our investigation to assess the combined performance of parameters. By integrating all the six parameters in various settings, we created $3^6 = 729$ unique configurations and studied their collective effects on test generation.

For each metric (i.e., line coverage, branch coverage, and number of faults), we developed separate tables to present the configurations' performances. These tables contrast the best-performing configurations against the default settings, facilitating direct comparisons. The findings, specifically in Tables 17 and 18, highlight that the optimal settings for achieving the highest line and branch coverage, at 61.34% and 27.40% respectively, were identical.

The analysis revealed a significant gap between the performances of the default and the best configurations, particularly in line and branch coverage, where the default settings ranked 538th and 565th respectively. This stark contrast, with around 9% and 6% differ-

Table 17 Line coverage results for the third set of experiments, including the top 15, the default, and the bottom 15 configurations (out of 729)

Rank	I	II	III	IV	V	VI	Line Coverage%
Best 15 Configurations							
1	NONE	3	0.5	0.2	0.8	0.9	61.34
2	NONE	1	0.5	0.5	0.2	0.9	59.29
3	NONE	3	0.5	0.5	0.8	0.5	58.40
4	FOCUSED	3	0.5	0.5	0.8	0.5	57.57
5	FOCUSED	5	0.3	0.2	0.8	0.1	56.97
6	NONE	3	0.3	0.2	0.8	0.5	56.95
7	LAST	1	0.3	0.2	0.8	0.5	56.87
8	NONE	5	0.3	0.8	0.2	0.5	56.60
9	NONE	5	0.3	0.2	0.8	0.5	56.42
10	NONE	5	0.1	0.5	0.5	0.5	56.24
11	FOCUSED	1	0.3	0.2	0.5	0.5	56.12
12	FOCUSED	3	0.3	0.2	0.8	0.5	56.09
13	FOCUSED	5	0.1	0.5	0.2	0.5	55.96
14	FOCUSED	3	0.5	0.8	0.5	0.9	55.89
15	NONE	1	0.5	0.2	0.8	0.1	55.87
Default Configuration							
538	LAST	5	0.5	0.5	0.5	0.9	52.09
Worst 15 Configurations							
715	LAST	5	0.1	0.8	0.5	0.9	49.31
716	LAST	1	0.3	0.8	0.8	0.5	49.28
717	LAST	3	0.5	0.5	0.2	0.5	48.82
718	LAST	5	0.1	0.8	0.8	0.9	48.62
719	LAST	3	0.5	0.8	0.8	0.5	48.49
720	LAST	5	0.3	0.8	0.2	0.1	48.45
721	FOCUSED	3	0.1	0.8	0.8	0.5	48.41
722	LAST	1	0.1	0.8	0.8	0.5	48.35
723	LAST	1	0.3	0.8	0.2	0.9	48.35
724	FOCUSED	5	0.1	0.8	0.8	0.5	48.34
725	NONE	5	0.1	0.8	0.8	0.5	48.09
726	LAST	3	0.1	0.8	0.8	0.5	47.87
727	NONE	1	0.3	0.8	0.8	0.5	47.80
728	LAST	5	0.5	0.8	0.5	0.5	47.36
729	LAST	5	0.1	0.8	0.8	0.5	46.19

ences in coverage, underlines the default settings' poor coverage compared to the best performing configuration.

The performance rankings for the configuration with the least line and branch coverage showed that the worst line coverage configuration was ranked 722nd for branch coverage, and the least effective branch coverage setting was almost the last for line coverage, positioned at 728th.

When focusing on fault detection, the default settings demonstrated slightly less inadequate performance, securing the 405th position out of 729, as per Table 19. This somewhat

Table 18 Branch coverage results for the third set of experiments, including the top 15, the default, and the bottom 15 configurations (out of 729)

I: feedbackDirectedSampling, II: maxSqlInitActionsPerMissingData, III: probOfApplySQLActionToCreateResources, IV: probOfRandomSampling, V: focusedSearchActivationTime, VI: baseTaintAnalysisProbability							
Rank	I	II	III	IV	V	VI	Branch Coverage%
Best 15 Configurations							
1	NONE	3	0.5	0.2	0.8	0.9	27.40
2	NONE	1	0.5	0.5	0.2	0.9	25.44
3	NONE	3	0.5	0.5	0.8	0.5	25.39
4	FOCUSED	3	0.5	0.5	0.8	0.5	25.15
5	NONE	5	0.1	0.5	0.5	0.5	25.03
6	FOCUSED	5	0.3	0.2	0.8	0.1	24.86
7	NONE	1	0.5	0.2	0.8	0.1	24.46
8	FOCUSED	1	0.1	0.8	0.8	0.1	24.39
9	FOCUSED	5	0.1	0.5	0.2	0.5	24.37
10	NONE	5	0.1	0.5	0.2	0.5	24.37
11	FOCUSED	5	0.1	0.8	0.8	0.1	24.15
12	FOCUSED	1	0.5	0.8	0.8	0.5	24.11
13	FOCUSED	3	0.3	0.2	0.2	0.9	24.08
14	NONE	3	0.3	0.2	0.8	0.5	24.06
15	NONE	5	0.3	0.2	0.2	0.9	24.06
Default Configuration							
565	LAST	5	0.5	0.5	0.5	0.9	21.09
Worst 15 Configurations							
715	FOCUSED	5	0.1	0.8	0.8	0.5	18.54
716	FOCUSED	1	0.1	0.2	0.2	0.9	18.37
717	FOCUSED	3	0.1	0.8	0.8	0.5	18.32
718	NONE	5	0.1	0.8	0.8	0.5	18.16
719	NONE	1	0.3	0.8	0.8	0.5	18.10
720	LAST	3	0.1	0.8	0.8	0.5	18.00
721	LAST	1	0.1	0.2	0.2	0.9	17.85
722	LAST	5	0.1	0.8	0.8	0.5	17.07
723	NONE	5	0.5	0.8	0.5	0.5	16.33
724	FOCUSED	1	0.1	0.8	0.8	0.5	16.28
725	NONE	1	0.1	0.8	0.8	0.5	16.18
726	FOCUSED	5	0.5	0.8	0.5	0.5	16.12
727	LAST	1	0.1	0.8	0.8	0.5	15.89
728	LAST	3	0.5	0.8	0.5	0.5	15.77
729	LAST	5	0.5	0.8	0.5	0.5	14.92

improved ranking suggests that the default settings were marginally more effective in this metric compared to their performance in line and branch coverage, though still far from optimal.

Table 20 not only compares the default, best, and worst configurations across the three metrics but also provides Vargha-Delaney and Mann-Whitney-Wilcoxon U-test analyses (Arcuri and Briand 2014). Following the detailed presentation of the three metrics, the fourth table we provide, Table 20, offers a further layer of analysis by comparing the

Table 19 Found faults results for the third set of experiments, including the top 15, the default, and the bottom 15 configurations (out of 729)

I: feedbackDirectedSampling, II: maxSqlInitActionsPerMissingData, III: probOfApplySQLActionToCreateResources, IV: probOfRandomSampling, V: focusedSearchActivationTime, VI: baseTaintAnalysisProbability							
Rank	I	II	III	IV	V	VI	Faults#
Best 15 Configurations							
1	FOCUSED	1	0.1	0.8	0.8	0.5	77.69
2	NONE	1	0.1	0.8	0.8	0.5	76.23
3	LAST	5	0.3	0.8	0.8	0.5	73.92
4	LAST	1	0.1	0.8	0.8	0.5	73.77
5	NONE	5	0.5	0.8	0.5	0.5	72.92
6	FOCUSED	3	0.5	0.8	0.8	0.5	72.46
7	FOCUSED	5	0.5	0.8	0.5	0.5	71.92
8	FOCUSED	5	0.5	0.8	0.8	0.5	71.71
9	LAST	3	0.5	0.8	0.5	0.5	71.69
10	FOCUSED	3	0.1	0.8	0.8	0.5	71.58
11	LAST	5	0.5	0.8	0.5	0.5	71.38
12	NONE	1	0.5	0.8	0.8	0.5	71.38
13	FOCUSED	5	0.1	0.8	0.8	0.5	71.33
14	NONE	3	0.5	0.8	0.8	0.5	71.31
15	NONE	3	0.1	0.8	0.8	0.9	71.21
Default Configuration							
405	LAST	5	0.5	0.5	0.5	0.9	60.87
Worst 15 Configurations							
715	FOCUSED	1	0.3	0.2	0.5	0.5	26.93
716	FOCUSED	5	0.5	0.8	0.5	0.1	26.93
717	LAST	3	0.5	0.8	0.5	0.1	26.86
718	LAST	1	0.3	0.2	0.5	0.5	26.29
719	NONE	3	0.5	0.8	0.5	0.1	26.21
720	LAST	3	0.3	0.2	0.5	0.5	26.14
721	LAST	3	0.1	0.5	0.2	0.5	23.58
722	LAST	3	0.1	0.5	0.2	0.9	21.83
723	FOCUSED	5	0.1	0.5	0.2	0.9	21.58
724	FOCUSED	3	0.1	0.5	0.2	0.9	21.42
725	LAST	1	0.1	0.5	0.2	0.9	21.42
726	NONE	1	0.3	0.5	0.2	0.9	21.17
727	NONE	3	0.1	0.5	0.2	0.9	20.58
728	LAST	5	0.1	0.5	0.2	0.9	20.42
729	NONE	5	0.1	0.5	0.2	0.9	20.42

default, best, and worst values for these metrics and employing the Vargha-Delaney effect size measure.

The Vargha-Delaney value for comparing line coverage between the worst and default settings revealed a value of 0.53 with a p-value of 0.85, suggesting a minor difference in line coverage. This result, while not statistically significant, indicates that the ‘worst’ settings may not necessarily reduce effectiveness compared to ‘default’ configurations. This challenges the traditional understanding of parameter impact, highlighting the need for a nuanced evaluation of tool performance beyond statistical significance. It suggests that con-

Table 20 The actual values for the Default, Worst, and Best Configurations along with their Vargha-Delaney comparison and *p-values* across the metrics

Metric	Default	Worst	Best	\hat{A}_{WD}	<i>p-val(WD)</i>	\hat{A}_{BD}	<i>p-val(BD)</i>
Line Coverage	52.09	46.19	61.34	0.53	0.85	0.54	0.76
Branch Coverage	21.09	14.92	27.40	0.46	0.71	0.56	0.60
Faults	60.87	20.42	77.69	0.42	0.49	0.58	0.50

figurations perceived as inferior may still be valuable under certain conditions, prompting a reevaluation of parameter optimization strategies in SBSE research and tool development.

5.6.1 Findings Summary for RQ4

Combining various parameter values significantly impacts EvoMASTER's effectiveness in REST API testing, especially in line and branch coverage. After analyzing 729 configurations, it became clear that optimal parameter combinations significantly outshine default settings. The best performing configurations attained 61.34% line and 27.40% branch coverage, marking an approximate increase of 9% in line coverage and 6% in branch coverage over the default configuration values. This suggests that careful tuning of parameters can lead to improved testing outcomes.

In terms of fault detection, the discrepancy between the best and default settings remains notable, with an average difference of approximately 17 faults. Interestingly, the default setting's ranking in fault detection exhibits a relative improvement, standing 405 out of 729. This is a marked contrast to its rankings for line and branch coverage, which are notably lower at 538 and 565 out of 729, respectively.

RQ4: Combining different parameter values significantly enhances code coverage and fault detection, outperforming default configurations. This demonstrates the potential of optimized parameter tuning to improve test generation outcomes.

5.7 Understanding the Impact of Parameters on the Test Generation

In this part of our research, we applied four distinct machine learning models, i.e., Random Forest Regressor, Gradient Boosting Regressor, Decision Tree Regressor, and Linear Regression, on the data from Section 5.6 to assess the influence of parameters on test generation metrics (i.e., line coverage, branch coverage, and found faults). The data frame comprised 729 rows, each representing averaged metric values across the 15 SUTs. Our methodology entailed evaluating each model's performance based on MSE and R^2 scores to identify the most effective model for each metric. The model with the best scores for each metric was then chosen for further analysis. Finally, the feature importance of the selected model was analyzed for each metric to determine the impact of the parameters. We used default hyperparameters for all models, with `random_state=42` for reproducibility. No explicit hyperparameter tuning was performed.

Table 21 Machine Learning Model Performance Evaluation Across the 3 Metrics

Model	Line Coverage		Branch Coverage		Faults	
	MSE	R ²	MSE	R ²	MSE	R ²
RandomForestRegressor	0.0002	0.3171	0.0001	0.3375	18.5165	0.7599
GradientBoostingRegressor	0.0002	0.4015	0.0001	0.3895	40.9707	0.4687
DecisionTreeRegressor	0.0003	-0.0976	0.0003	-0.1883	47.5193	0.3838
LinearRegression	0.0003	-0.0195	0.0002	-0.0480	64.7548	0.1603

Table 22 Feature Importance for Automated Test Generation

Feature	Feature Importance (out of 1)		
	Line Coverage	Branch Coverage	Faults
feedbackDirectedSampling	0.5535	0.5651	0.0526
probOfRandomSampling	0.1360	0.0842	0.2062
focusedSearchActivationTime	0.1300	0.0863	0.1786
baseTaintAnalysisProbability	0.1105	0.1876	0.2003
probOfApplySQLActionToCreateResources	0.0353	0.0368	0.2209
maxSqlInitActionsPerMissingData	0.0347	0.0401	0.1415

5.7.1 Impact of Parameters on Line Coverage

For the line coverage metric, both the Decision Tree Regressor and Linear Regression models yielded negative R² values which indicate poor performance of the regression models (Chicco et al. 2021).

Among the remaining models, the Gradient Boosting Regressor demonstrated the lowest MSE and the highest R² (see Table 21). However, the R² value of 0.4015 suggests that while the model captures some variance in the data, a significant portion remains unexplained. Despite this limitation, the model identified `feedbackDirectedSampling` as the most influential parameter, with a high importance score of 0.5535 (out of 1). Parameters such as `probOfRandomSampling` (0.1360), `focusedSearchActivationTime` (0.1300), and `baseTaintAnalysisProbability` (0.1105), while less significant, still play important roles in the model's performance. The normalized feature importance values for line coverage are shown in Table 22. By evaluating all four machine learning models, we found that the Gradient Boosting Regressor provided the best balance of MSE and R², though its predictive power is somewhat limited.

5.7.2 Impact of Parameters on Branch Coverage

Similar to the line coverage analysis, all four models were applied to the dataset. As shown in Table 21, our evaluation identified the Gradient Boosting Regressor as the best-performing model, achieving the lowest MSE and the highest R². However, with an R² value of 0.3895, the model explains only a portion of the variance in branch coverage, suggesting that additional factors may influence the results. Despite this limitation, the findings highlight `feedbackDirectedSampling`, with its significant importance score of 0.5651, as the most crucial factor in augmenting the branch coverage of tests generated by EvoMASTER. The next most important parameter, albeit to a much lesser extent, is shown to be `baseTaintAnalysisProbability`, with a score of 0.1876.

5.7.3 Impact of Parameters on Fault Detection

For the analysis of faults, unlike the previous two metrics, the Random Forest Regressor was distinguished as the superior model by demonstrating the most advantageous balance between MSE and R² scores (Table 21). This model, as displayed in Table 22, pinpointed `probOfApplySQLActionToCreateResources` with importance score of 0.2209, `probOfRandomSampling` with 0.2062, and `baseTaintAnalysisProbability` with 0.2003 as pivotal parameters influencing fault detection.

5.7.4 Aggregated Analysis of Parameter Impact

In this part of the research, we undertook an analysis to pinpoint the most influential parameters in optimizing automated test generation. To achieve this, we followed a detailed plan of combining and simplifying data from two different models: one for the line and branch coverage (i.e., Gradient Boosting Regressor), and another for the found faults (i.e., Random Forest Regressor).

In `scikit-learn`,⁴ the library we used to apply machine learning models, feature importance for tree-based models like Random Forest and Gradient Boosting is computed based on the reduction in the impurity criterion across all trees in the ensemble. For Random Forest Regressor and Gradient Boosting Regressor, the default criterion in `scikit-learn` is defined as `squared_error` and `friedman_mse`, respectively (`scikit-learn` developers 2025a, b). The former is in fact the MSE and the latter is another variation of MSE which accounts for interaction effects between features, improving performance for gradient boosting models (Friedman 2001). For Random Forest Regressor, the importance of a feature is calculated as the total reduction in impurity caused by that feature, averaged across all decision trees in the forest. For Gradient Boosting Regressor, the importance is calculated similarly but considers the sequential nature of the model. It sums the impurity reduction for each feature across all trees and normalizes the values so that the total importance equals one. It is worth noting that impurity-based feature importance can be biased towards high cardinality features (i.e., features with a higher number of unique values), however, this does not apply to this research (`scikit-learn` developers 2025a, b).

The outcome of this analytical process, as shown in Table 22, identified `feedbackDirectedSampling` as the most influential parameter for line coverage and branch coverage, with importance scores of 0.5535 and 0.5651, respectively. However, since these importance values were derived from the Gradient Boosting Regressor, which exhibited moderate R² values (0.4015 for line coverage and 0.3895 for branch coverage), this conclusion should be interpreted with caution, as the model does not fully explain the variance in the data. Additionally, `feedbackDirectedSampling` did not appear to have a significant impact on the number of detected faults, with a low importance score of 0.0526. Instead, `probOfApplySQLActionToCreateResources` had the highest impact on fault detection, with an importance score of 0.2209, a result derived from the Random Forest Regressor, which demonstrated a considerably stronger R² of 0.7599.

⁴ <https://scikit-learn.org/1.5/index.html>

5.7.5 Findings Summary for RQ5

To answer this research question, a comprehensive analysis was conducted through an evaluation of machine learning models and their parameter importance features. Four distinct models, including Random Forest Regressor, Gradient Boosting Regressor, Decision Tree Regressor, and Linear Regressor, were utilized for assessing parameter influence on test generation metrics (i.e., line coverage, branch coverage, and found faults).

Our methodology involved selecting the most appropriate model for each metric by evaluating model performance based on MSE and R^2 scores, followed by an analysis of their feature importance scores. As presented in Table 22, `feedbackDirectedSampling` was identified as the most impactful parameter for line and branch coverage, with feature importance scores of 0.5535 and 0.5651, respectively; however, this should be interpreted cautiously due to moderate R^2 values. Conversely, `feedbackDirectedSampling` had minimal impact on fault detection, while `probOfApplySQLActionToCreateResources` emerged as the most influential parameter, achieving an importance score of 0.2209. Other parameters, including `baseTaintAnalysisProbability`, `probOfRandomSampling`, and `focusedSearchActivationTime`, also exhibited notable influence, though to varying extents across different metrics.

RQ5: Our analysis reveals that `feedbackDirectedSampling` was the most influential for code coverage but with moderate reliability, while `probOfApplySQLActionToCreateResources` had the strongest impact on fault detection with higher model confidence.

5.8 Results of the Fourth Set of Experiments

In the fourth set of experiments, we focused on validating the effectiveness of the best parameter settings identified for line coverage, branch coverage, and found faults against the default settings. This validation was achieved through a more rigorous testing process, where each one of the selected configurations, including the default, were subjected to 20 repetitions under a one-hour time budget. Notably, the previous set of analysis revealed that the best settings for line and branch coverage were identical which left us having three distinct configurations instead of four. The results of this set of experiment, which took nearly 3 days (when running 15 experiments in parallel), are presented in Table 23.

Table 23 includes the parameter settings for the best coverage, best faults, and the default alongside their achieved outcome for the three metrics and their respective ranking. For line and branch coverage, the best configuration resulted in 55.21% line coverage and 22.79% branch coverage, both ranking first, as expected. This configuration also led to the detection of 60.81 faults which makes it the second best setting in that metric. In contrast, the configuration optimized for detecting the highest number of faults notably achieved the highest number of faults with 65.96, while slightly trailing in line and branch coverage with 54.68% and 22.66%, respectively.

The default settings yielded 52.69% line coverage and 21.29% branch coverage, both ranking third. Additionally, this configuration detected 60.25 faults, also ranking third among the configurations tested.

Table 23 Results of running the *Best Faults*, *Best Coverage*, and *Default* 20 times with 1 hour time budget

Parameters		I: feedbackDirectedSampling, II: maxSqlInitActionsPerMissingData,	III: probOfApplySQLActionToCreateResources, IV: probOfRandomSampling,	V: focusedSearchActivationTime, VI: baseTaintAnalysisProbability	X: Line Coverage%, Y: Branch Coverage%, Z: Faults#								
Metrics	Title	I	II	III	IV	V	VI	X	#	Y	#	Z	#
Best Coverage	NONE	3	0.5	0.2	0.8	0.9	0.521	1	22.79	1	60.81	2	
Best Faults	FOCUSED	1	0.1	0.8	0.8	0.5	54.68	2	22.66	2	65.96	1	
Default	LAST	5	0.5	0.5	0.5	0.9	52.69	3	21.29	3	60.25	3	

These results show the difference that fine-tuning EVOMASTER's parameters can make in the performance of test generation. The best configurations for coverage and fault detection consistently outperformed the default settings, affirming the hypothesis that optimized parameter combinations can lead to substantial improvements in REST API testing performance.

Among the three evaluated configurations, the *Best Faults* configuration emerges as the most effective. Not only does it surpass the default setting, but it also significantly exceeds the *Best Coverage* configuration in fault detection. Furthermore, the trade-off for its superior fault detection capability is minimal, resulting in slightly reduced branch and line coverage, both by less than 1%.

5.8.1 Findings Summary for RQ6

In our exploration to identify an optimal parameter configuration for test generation, we discovered that optimized parameter settings significantly enhance performance over default configurations. Our rigorous examination involved 20 repetitions under a one-hour time budget, conclusively demonstrating that these optimized configurations achieved superior results in line coverage, branch coverage, and fault detection. Notably, the configuration optimized for fault detection not only surpassed the default and *Best Coverage* settings in its primary objective by detecting an average of 65.96 faults, compared to nearly 60 faults detected by both the default and *Best Coverage* settings, but also maintained competitive coverage metrics, lagging less than one percent behind the *Best Coverage* in both line and branch coverage. This comprehensive analysis underscores the pivotal role of parameter tuning in improving test generation outcomes, affirming the existence of a "best" configuration that excels in fault detection while sustaining worthy coverage performance.

Note that, from a theoretical standpoint, no search algorithm can be best on all possible problems (Wolpert and Macready 1997). However, for specific domains (e.g., test data generation), there can be specialized algorithms that give better results on average. This is especially the case for SBSE tools that aim to very specific domains (e.g., REST API testing in the case of EVOMASTER).

A further note to consider is that, among the evaluated 729 configurations, some of them might give better results. The identification of such better configurations might have been lost in the noise introduced by randomness when running experiments only once. However, the point is not to find the best possible configuration, but whether to find out if a re-tuning of an existing SBSE tool can lead to significantly better results of practical value.

RQ6: Optimized configurations outperform default settings in test generation, notably a configuration optimized for fault detection, which surpasses others in fault detection while maintaining high coverage metrics. This confirms the existence of a significantly better configuration compared to default settings.

5.9 Results of the Fifth Set of Experiments

In the fifth and final set of experiments, we aimed to conduct an in-depth comparison between the best setting identified in the previous step (i.e., *Best Faults* from Subsection 5.8) and the default setting by allocating a generous 10-hour time budget for each run over five repeti-

tions. The experiment took around 4 days (when running 15 jobs in parallel), following the formula $2 \times 15 \times 5 \times 10\text{h} \div 15 \div 24\text{h} = 4\text{d}$.

Table 24 presents the results of this set of experiment, showcasing the performance metrics for line coverage, branch coverage, and found faults across 15 SUTs. The values in the table are the result of the Vargha-Delaney effect size measure (Vargha and Delaney 2000) of the best configuration compared to that of the default. For each SUT, Table 24 presents the outcomes for both the best and default settings followed by the mean and median values for line coverage, branch coverage, and fault detection, alongside D'Agostino's K-Squared test (D'agostino et al. 1990) for skewness which the latter provides insights into the distribution of the results.

The mean values, 0.59 for line coverage, 0.55 for branch coverage, and 0.66 for faults, suggest the configuration picked from the previous rounds still outperforms the default setting across the SUTs. The application of D'Agostino's K-squared test (D'agostino et al. 1990) to our experimental results reveals nuanced insights into the skewness of distributions for the three metrics. As shown in Table 24, with skewness values of -0.7 for line coverage, -0.24 for branch coverage, and 0.24 for faults, the analysis indicates a slight negative skewness for coverage metrics and a minor positive skewness for fault detection. This suggests that while most SUT instances trended towards higher performance levels in coverage, the fault detection outcomes were more dispersed, with a general tendency towards lower fault counts albeit with some instances identifying a significantly higher number of faults. However, the associated p-values for these skewness metrics, 0.17 for line coverage, and 0.63 for both branch coverage and faults, denote that these deviations from symmetric distribution do not statistically compromise the integrity of our conclusions. Thus, despite the observable distribution tendencies in our data, these patterns do not significantly detract from the overall validity of our analytical interpretations, ensuring the reliability of our experimen-

Table 24 Vargha-Delaney analysis of the *Best Faults* versus the default configuration after 5 repetitions with 10 hour time budget

SUT	Line Coverage	Branch Coverage	Faults
catwatch	0.60	0.64	0.88
cwa-verification	0.52	0.40	0.60
features-service	0.55	0.73	0.75
genome-nexus	0.24	0.24	0.36
gestaohospital-rest	0.60	0.40	0.50
ind0	0.92	0.90	0.98
languagetool	0.56	0.22	0.61
market	0.38	0.42	0.70
ocvn-rest	0.78	0.82	1.00
proxyprint	0.00	0.00	0.40
rest-ncs	0.70	0.60	0.50
rest-news	0.68	0.54	0.50
rest-scs	1.00	1.00	0.80
restcountries	0.70	0.70	0.50
scout-api	0.64	0.58	0.84
Mean	0.59	0.55	0.66
Median	0.60	0.58	0.61
D'Agostino's K ²	-0.70	-0.24	0.24
p-value	0.17	0.63	0.63

tal analysis. In other words, the deviations indicated by the K^2 values are not statistically significant.

5.9.1 Time Based Analysis of the Results Across SUTs

The results of the fifth set of experiments are presented by detailed time-based graphs across all the 15 SUTs and their aggregated averages. They reveal insightful patterns regarding the efficacy of the *Best Faults*, chosen as the best configuration, versus the default configurations over a 10-hour testing period. These patterns are visually depicted in Figs. 1, 2, and 3 for line coverage, branch coverage, and fault detection metrics, respectively. Each figure is consisting of 16 graphs, including 15 graphs per SUT along with an average.

Overall, the analysis demonstrates that the best configuration consistently outperforms the default setting across the majority of SUTs, particularly when evaluating the aggregated performance over time. This trend is especially witnessed in the average graphs which indicates a clear advantage of the optimized configurations in enhancing test generation effectiveness.

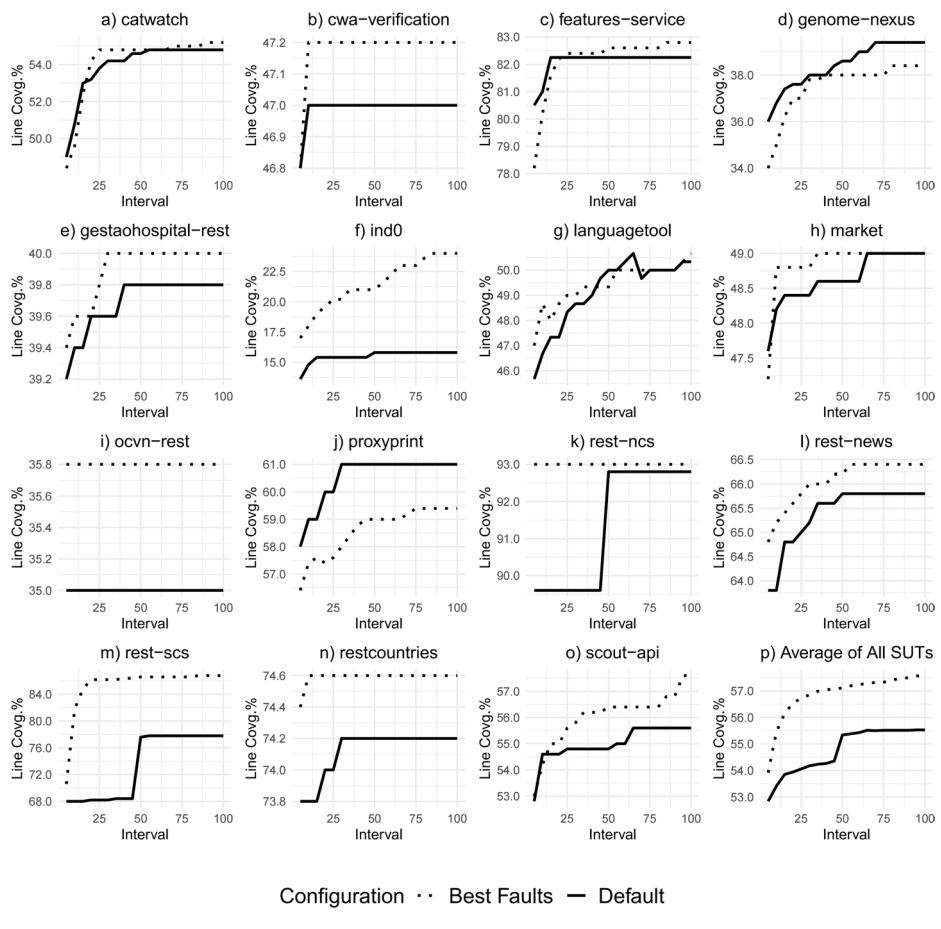
However, exceptions were noted in specific SUTs, such as `genome-nexus` and `proxy-print`, where the default configuration slightly surpassed the best setting across all three metrics. For `genome-nexus`, the performance difference was minimal, with line coverage around 1%, branch coverage even less than 1%, and fault detection differing by fewer than 1 fault on average. Similarly, `proxy-print` showed a slight advantage of the default setting that surpassed the best setting by fewer than 5 faults.

Additionally, `language-tool` showed an interesting pattern where the default settings slightly outperformed the best configuration in terms of line coverage, even though the best configuration had an early advantage. The crossover occurred at approximately the interval of 30% of the time budget (i.e., 10 hours), with the default maintaining a slim lead of less than 1% by the end of the testing period.

Conversely, other SUTs such as `market` displayed a different dynamic, where the best configuration, even if initially lagging behind, finally achieved superior outcomes. This evaluation underscores the complexity of optimizing test generation configurations and emphasizes the importance of considering SUT-specific factors in achieving the most effective testing outcomes.

5.9.2 Findings Summary for RQ7

In the final set of experiments, the *Best Faults* configuration, identified previously as superior, was evaluated against default settings across 15 SUTs over a 10-hour testing budget. Results, highlighted in Table 24 and Figs. 1, 2, and 3, confirm the *Best Faults*' dominance in improving line coverage, branch coverage, and fault detection. While mean values across metrics showed clear superiority of *Best Faults*, some exceptions like `genome-nexus` and `proxy-print` where defaults performed marginally better were witnessed. This illustrates the complexity of SUT-specific responses to parameter tuning.



Configuration ··· Best Faults — Default

Fig. 1 Achieved line coverage of the 15 SUTs over time based on the fifth set of experiments and their average

RQ7: Best Faults configuration outperformed the default setting in comprehensive testing, with some SUT-specific exceptions in which the default setting had marginal advantage.

6 Discussion

In Section 5.7.4, it was detected that the most impactful parameters for code coverage and fault detection are different. This raises the question of whether running multiple rounds of test generation, each optimized for different objectives, could lead to improved overall results. One possible strategy could be to execute separate test generation rounds using the best configuration for each metric, followed by unifying the generated test cases. While this

approach could potentially maximize both code coverage and fault detection, it also introduces computational overhead, which may not always be justified by the gains.

Another important consideration is the performance of default versus optimized configurations, as observed in Section 5.9.1. While optimized configurations generally outperformed the default settings across most SUTs, there were instances where the default performed either comparably well or even better (e.g., `genome-nexus` and `proxy-print`). This suggests that parameter tuning effectiveness may be context-dependent, and that using a fixed optimal configuration may not always yield the best results across different experimental subjects.

It was also observed that the most impactful parameter for both line coverage and branch coverage is the same, while the rankings of the remaining parameters differ. At this stage, we do not have any clear hypothesis to explain this ranking discrepancy. Moreover, the distinction between parameters that improve code coverage and those that enhance fault detection is indeed important to scrutinize. Currently, we do not have a clear explanation for this either. However, this could stem from the fact that high coverage does not necessarily translate to exposing more faults. While some parameters prioritize maximizing coverage, fault detection often benefits from generating diverse, extreme or unexpected test inputs that trigger failures. Investigating these effects further could provide deeper insights into how parameter tuning impacts different aspects of SBSE-driven test generation.

One important consideration when evaluating the effectiveness of parameter tuning is whether the computational cost of identifying the best-performing configurations is justified by the improvements they offer. While optimized configurations generally outperform default settings, the observed gains vary across different SUTs, and in some cases, the improvements may not be substantial. The process of identifying the best-performing configurations in this study was indeed computationally costly, making it impractical to run such experiments every time a new test generation is performed. However, rather than tuning parameters for each individual case, such an optimization process can be conducted periodically to account for tool evolution and changing software characteristics. By periodically re-evaluating and updating parameter settings, it is possible to maintain strong performance while avoiding the need for frequent, resource-intensive tuning. Further research could explore the balance between tuning frequency and performance stability, ensuring that test generation remains both efficient and effective over time.

In this study, we have analyzed the impact of parameter tuning in an evolving SBSE tool, throughout the years and different studies. We have empirically shown that parameter re-tuning can be beneficial, especially when comparing with the results of old studies. Still, the question remains of why is this the case? What are the conditions that lead to the need for re-tuning of parameters? Answering these questions might lead to obtain insight that could be used to define strategies to predict when and how re-tuning should be done to obtain best final performance.

Different results compared to existing studies could be explained with at least four different reasons:

- (a) the original studies were conducted with methodological errors.
- (b) new values for parameters were introduced, so what used to be the best, is now outperformed by the new choice.

- (c) the tool received bug fixes which influenced the effects of some of the parameters.
- (d) the gradual improvement of the tool, without major disruptive effects, led to changes of effects of the parameters.

Investigating those hypotheses would require new sets of experiments, which are not trivial to do. For example, for point (c) and (d), currently a mature tool such as EvoMASTER has more than 11 000 code commits.⁵ While techniques like commit bisection can reduce the number of versions to analyze in case of drastic breaking changes, such clear breakpoints did not exist in our case, as changes appeared to be gradual. Our experiments in this paper on a single version of EvoMASTER took 922 days. Even if using a smaller number of different configurations, investigating hundreds or thousands of different versions of EvoMASTER would be computationally expensive. And, even if a specific commit could be identified in which a drastic change of performance was introduced, then there would still need to have in-depth analyses to understand how the changed code impacted the results. With a large codebase of more than 200 000 lines of code, and code commits that can be in the order of thousands of changed lines, these investigations would not be cheap. These are important research questions that will need to be investigated in future work.

7 Threats to Validity

Conclusion Validity. Acknowledging the inherent randomness in SBSE methods, we carried out each experiment multiple times to ensure reliability. Specifically, we conducted 10, 10, 20, and 5 repetitions for the first, second, fourth, and fifth sets, respectively. An exception was made for the third set of experiments, where the extensive array of configurations, totaling 729, necessitated a different approach. To ensure the conclusions drawn are reliable, statistical analysis methods were applied, such as the Mann-Whitney-Wilcoxon U-tests (*p-value*) and the Vargha-Delaney effect sizes (\hat{A}_{12}), adhering to common software engineering research guidelines (Arcuri and Briand 2014).

Regarding the parameter impact analysis, we evaluated a variation of four machine learning models and picked the one with the best evaluation score (i.e., MSE and R^2) for each metric. A potential threat to the validity of our findings is the risk of overfitting of the chosen machine learning models to the specific SUTs. To hinder this, our selection of SUTs in the EMB, along with an industrial SUT, showcases a diversity across multiple dimensions: they span various domains (e.g., healthcare, e-commerce), are built on different Java frameworks (e.g., Spring, DropWizard), incorporate a wide range of features such as database and authentication support, and according to Table 2, vary greatly in size (i.e., number of source files, lines of code and endpoints). This deliberate variety aims to mitigate the risk of overfitting and strengthen the generalizability of our research outcomes.

Additionally, we utilized D'Agostino's K-squared test in the fifth set of experiments to assess the skewness of our results' distribution, ensuring a deeper understanding of our

⁵ <https://github.com/WebFuzzing/EvoMaster>

data's symmetry. The test results affirm that the observed skewness does not significantly impact the reliability of our findings.

Internal Validity. The EvoMASTER tool and the EMB case studies, with the exception of one industrial API, are publicly accessible. Being the developers of EvoMASTER, we also shared the scripts for experiment setup from our previous studies. These scripts served as the foundation for configuring our current experiments. The adapted scripts have been made available on the EvoMASTER GitHub repository (EvoMaster 2025), facilitating study replication and review.

External Validity. This research was designed with a focus on evolutionary algorithms within search-based software testing, involving REST APIs and the EvoMASTER tool. The results may not be generalizable to other software types, tools, algorithms, or domains. Nonetheless, to broaden the scope of case studies, we included all JVM-based REST APIs from the EMB corpus aimed at Web/Enterprise API testing and an industrial REST API, encompassing 15 REST APIs, 318k LOC, and 542 endpoints. It is possible that selecting different parameters could yield varied outcomes. Given the significant time required for experiment runs on system test generation (for instance, 61 days in this study, facilitated by parallel execution), our investigation was constrained to a selected set of parameters. This limitation leaves room for future research to explore the impact of additional parameters. Additionally, the study was performed with a limited variation of search budgets (i.e., 1-hour for the first 3 sets, and 10-hour for the fifth set), acknowledging that varying search budgets might influence the tuning efficacy. Specifically, a shorter search budget may favor configurations that prioritize immediate search space ‘exploitation’ over broader ‘exploration’, which demands more time.

8 Conclusion and Future Work

The main goal of this study was to widen our approach to parameter tuning in SBSE through conducting several experiments on the selected tool (i.e., EvoMASTER) which has been extended and developed over time across multiple studies and discover the impact of the parameter tuning on the test generation performance. Our initial experiments show that, although some parameters from previous studies remain effective despite significant updates and the integration of various techniques into the tool over the years, the tool’s continuous evolution necessitates a reassessment of other parameters. This underscores the significance of periodic re-evaluation of parameter settings to ensure the sustained relevance and efficiency of SBSE tools. Furthermore, the incorporation of a wider collection of SUTs not only reinforces the external validity of our results, but also affirms the robustness of previous conclusions across a broader spectrum of testing scenarios. We also explored a more diverse set of values for the numerical parameters; however, no new optimal configurations were found beyond those already discovered in the first set of experiments. Furthermore our analysis demonstrates a notable enhancement in test generation outcomes when it comes to the combination of the different parameter values. This suggests that parameter optimization can substantially improve line coverage, branch coverage, and fault detection.

Regarding the impact of parameters on test generation metrics, our study identifies `feedbackDirectedSampling` as the most influential parameter on code coverage (i.e., line and branch coverage) through applying machine learning algorithms. However, since this conclusion is drawn from models with moderate R^2 values, it should be interpreted with caution. In contrast, `probOfApplySQLActionToCreateResources` was found to be the most impactful parameter for fault detection, a result supported by a model with stronger predictive power. Future work could explore the use of tuning frameworks such as `irace` (López-Ibáñez et al. 2016) are designed for algorithm configuration problems with stochastic and expensive evaluations.

The rigorous evaluation of optimized parameter configurations against default settings further emphasizes the advantages of parameter tuning. Notably, the configuration which yielded the highest number of faults emerges as a superior choice while maintaining competitive performance in code coverage metrics.

Moreover, our exploration into the longitudinal performance of the best parameter settings shows that the configuration identified as the best generally surpasses the default setting. However, the varied behavior of specific SUTs highlights the need for context-aware parameter tuning. This insight emphasizes the value of considering SUT-specific characteristics in the optimization process to achieve the most effective testing outcomes.

As this study illuminates the dynamic nature of parameter tuning in SBSE and its substantial impact on test generation effectiveness, it opens a dialogue for other researchers and practitioners about the practicalities of implementing these insights. While the benefits of retuning are shown, finding the optimal frequency for such adjustments is a complex challenge, especially considering the resource-intensive nature of the required experiments. Our study alone, executed in parallel, spanned a total of 62 days. This highlights the need for a strategic approach to retuning which requires balancing between achieving the maximum capabilities out of testing tools and the significant resources required. Future work could explore methodologies for predicting when a retuning might be most beneficial or develop more cost-effective strategies for keeping parameter configurations at peak performance without the need for exhaustive reevaluation.

In conclusion, our study not only reaffirms the importance of ongoing parameter tuning for SBSE evolving tools, but also broadens the understanding of parameter interaction effects. By highlighting the potential for optimization through careful parameter combination and adjustment, we pave the way for future research endeavors aimed at enhancing the precision and efficiency of automated testing tools in adapting to the changing landscape of software development. In response to our findings, we have applied the newly identified optimal parameter values in `EvoMASTER` by updating the default values in the `EMConfig.kt` file for the 6 tuned parameters in this study. These are now the new “default” settings in `EvoMASTER`.

Appendix A: Tables for the Second Set of Experiments

In the following tables, the highest value for each SUT is highlighted in bold, as well as the Average and Median rows. The default parameter value is underlined in the row titles.

Table 25 Line Coverage% performance for `maxSqlInitActionsPerMissingData`

SUT	1	3	<u>5</u>	7	15	20
catwatch	50	50	50	52	49	51
features-service	80	81	81	81	80	80
proxyprint	53	52	53	54	52	54
rest-news	64	65	64	64	64	65
scout-api	53	52	52	53	53	52
Average (original)	60	60	60	61	60	60
Avg Std Dev (original)	2	2	2	2	2	2
Median (original)	53	52	53	54	53	54
95% CI-LB (original)	58	59	59	59	58	59
95% CI-UB (original)	62	62	61	63	62	62
cwa-verification	47	47	47	47	47	47
genome-nexus	36	34	35	34	34	36
gestaohospital-rest	40	39	40	40	40	40
ind0	18	17	15	16	16	17
languagetool	47	36	40	36	35	34
market	48	47	47	48	47	48
ocvn-rest	35	35	35	35	35	35
rest-ncs	92	92	91	93	92	92
rest-scs	79	78	75	74	78	75
restcountries	70	71	72	72	70	71
Average (others)	51	50	50	50	49	50
Avg Std Dev (others)	6	2	3	2	2	2
Median (others)	47	43	44	44	44	44
95% CI-LB (others)	47	48	48	48	48	48
95% CI-UB (others)	55	51	51	51	51	51
Average (all)	54	53	53	53	53	53
Avg Std Dev (all)	5	2	3	2	2	2
Median (all)	50	50	50	52	49	51
95% CI-LB (all)	51	52	52	51	51	52
95% CI-UB (all)	57	55	55	55	54	55
Max Count	9	4	5	10	4	7

Table 26 Branch Coverage% performance for maxSqlInitActionsPerMissingData

SUT	1	3	5	7	15	20
catwatch	19	19	19	20	18	19
features-service	25	26	24	26	25	25
proxyprint	12	12	13	14	11	13
rest-news	23	24	24	23	24	24
scout-api	23	23	23	23	23	23
Average (original)	20	21	21	21	20	21
Avg Std Dev (original)	2	1	1	2	1	1
Median (original)	23	23	23	23	23	23
95% CI-LB (original)	19	20	20	20	19	20
95% CI-UB (original)	22	22	21	22	21	22
cwa-verification	9	9	9	9	9	9
genome-nexus	19	18	18	18	18	19
gestaohospital-rest	3	3	3	3	3	3
ind0	3	3	2	2	3	3
languagetool	18	14	16	14	14	13
market	4	3	3	3	3	3
ocvn-rest	6	6	6	6	6	6
rest-ncs	66	66	65	66	66	66
rest-scs	44	45	41	40	44	41
restcountries	56	56	56	56	56	56
Average (others)	23	22	22	22	22	22
Avg Std Dev (others)	3	2	2	2	1	1
Median (others)	14	12	12	12	12	11
95% CI-LB (others)	21	21	21	21	21	21
95% CI-UB (others)	25	23	23	23	23	23
Average (all)	22	22	21	22	22	22
Avg Std Dev (all)	2	2	2	2	1	1
Median (all)	19	18	18	18	18	19
95% CI-LB (all)	20	21	21	20	20	20
95% CI-UB (all)	24	23	22	23	23	22
Max Count	10	10	6	9	8	9

Table 27 Faults# performance for maxSqlInitActionsPerMissingData

SUT	1	3	5	7	15	20
catwatch	26	26	27	27	26	26
features-service	32	33	32	32	32	32
proxyprint	86	81	85	84	86	86
rest-news	8	8	7	7	8	7
scout-api	92	90	87	89	89	89
Average (original)	49	48	48	48	48	48
Avg Std Dev (original)	3	2	3	2	3	2
Median (original)	32	33	32	32	32	32
95% CI-LB (original)	46	46	46	46	46	46
95% CI-UB (original)	51	49	49	49	51	50
cwa-verification	8	8	8	8	8	8
genome-nexus	20	20	19	20	20	21
gestaohospital-rest	22	22	22	22	22	22
ind0	45	43	43	42	44	45
languagetool	4	5	7	4	4	4
market	19	19	20	20	20	20
ocvn-rest	533	534	529	538	528	533
rest-ncs	6	6	6	6	6	6
rest-scs	9	7	7	8	9	7
restcountries	2	2	2	2	2	2
Average (others)	67	67	66	67	66	67
Avg Std Dev (others)	2	3	3	2	2	2
Median (others)	14	14	14	14	14	14
95% CI-LB (others)	65	65	65	66	65	65
95% CI-UB (others)	68	68	68	68	68	69
Average (all)	61	60	60	61	60	61
Avg Std Dev (all)	2	2	3	2	2	2
Median (all)	20	20	20	20	20	21
95% CI-LB (all)	59	58	59	59	59	59
95% CI-UB (all)	62	62	62	62	62	62
Max Count	9	6	7	7	8	8

Table 28 Line Coverage% performance for probOfApplySQLActionToCreateResources

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	46	51	52	50	49	50	50	50	50	51	49
features-service	80	81	81	81	81	81	81	81	81	81	82
proxyprint	50	53	53	52	52	53	55	52	53	52	53
rest-news	62	64	64	64	64	64	64	64	64	64	64
scout-api	53	52	53	52	52	52	52	52	52	53	52
Average (original)	58	60	61	60	60	60	60	60	60	60	60
Avg Std Dev (original)	3	2	3	2	2	2	2	2	2	2	2
Median (original)	53	53	53	52	52	53	55	52	53	53	53
95% CI-LB (original)	57	58	58	58	58	59	59	58	59	59	59
95% CI-UB (original)	60	62	63	61	61	61	62	62	62	62	61
cwa-verification	47	47	47	47	47	47	47	47	47	47	47
genome-nexus	36	34	32	35	35	35	34	35	35	34	35
gestaohospital-rest	39	39	39	39	40	40	39	39	40	40	39
ind0	15	14	14	15	15	15	16	17	15	16	19
languagetool	36	36	42	36	35	40	39	35	34	35	34
market	48	48	48	48	48	47	48	47	48	48	48
ocvn-rest	35	35	35	35	35	35	35	35	35	35	35
rest-ncs	92	93	92	92	92	91	91	93	92	92	92
rest-scs	78	78	79	80	77	75	75	79	76	78	81
restcountries	70	71	72	70	70	72	70	72	70	70	70
Average (others)	50	50	50	50	49	50	49	50	49	50	50
Avg Std Dev (others)	2	3	3	2	2	3	2	2	2	2	2
Median (others)	43	43	44	43	44	44	43	43	44	44	43
95% CI-LB (others)	48	48	48	48	48	48	48	48	47	48	48
95% CI-UB (others)	51	51	52	52	51	51	51	52	51	51	52
Average (all)	52	53	54	53	53	53	53	53	53	53	53
Avg Std Dev (all)	2	2	3	2	2	3	2	2	2	2	2
Median (all)	48	51	52	50	49	50	50	50	50	51	49
95% CI-LB (all)	51	51	51	51	51	52	51	51	51	51	52
95% CI-UB (all)	54	55	55	55	54	55	55	55	54	55	55
Max Count	5	5	8	4	5	5	5	5	5	6	7

Table 29 Branch Coverage% performance for probOfApplySQLActionToCreateResources

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	17	19	20	19	19	19	19	19	19	20	19
features-service	24	24	24	24	25	24	25	25	24	24	25
proxyprint	11	12	13	12	11	13	14	12	13	13	14
rest-news	21	22	22	23	23	24	24	24	24	24	24
scout-api	24	22	23	22	23	23	23	23	22	23	22
Average (original)	19	20	20	20	20	21	21	21	20	21	21
Avg Std Dev (original)	1	2	2	1	1	1	1	1	1	2	1
Median (original)	21	22	22	22	23	23	23	23	22	23	22
95% CI-LB (original)	18	19	19	19	19	20	20	19	19	19	20
95% CI-UB (original)	20	21	22	21	21	21	22	21	22	22	22
cwa-verification	9	9	9	9	9	9	9	9	9	9	9
genome-nexus	19	17	15	18	18	18	18	18	18	18	18
gestaohospital-rest	3	3	3	3	3	3	3	3	3	3	3
ind0	3	2	2	3	3	2	3	3	2	2	3
languagetool	14	14	16	14	13	16	16	13	13	14	14
market	4	4	4	3	4	3	4	3	4	3	4
ocvn-rest	6	6	6	6	6	6	6	6	6	6	6
rest-ncs	66	66	66	66	65	65	65	66	65	66	66
rest-scs	43	44	45	46	43	41	43	45	42	43	46
restcountries	56	56	56	55	56	56	55	56	55	55	56
Average (others)	22	22	22	22	22	22	22	22	22	22	22
Avg Std Dev (others)	2	2	2	2	2	2	2	2	2	2	2
Median (others)	12	12	12	12	11	12	12	11	11	12	12
95% CI-LB (others)	21	21	21	21	21	21	21	21	20	21	21
95% CI-UB (others)	23	23	24	24	23	23	23	24	23	23	24
Average (all)	21	21	22	22	21	21	22	22	21	22	22
Avg Std Dev (all)	2	2	2	2	2	2	2	2	2	2	1
Median (all)	17	17	16	18	18	18	18	18	18	18	18
95% CI-LB (all)	20	20	20	20	20	21	21	21	20	20	21
95% CI-UB (all)	22	23	23	23	23	22	23	23	22	23	23
Max Count	9	6	8	6	7	6	9	8	5	6	11

Table 30 Faults# performance for probOfApplySQLActionToCreateResources

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	25	26	26	27	27	27	26	26	25	26	27
features-service	32	33	33	33	33	32	34	33	32	33	34
proxyprint	84	82	86	86	86	85	83	83	84	83	84
rest-news	8	8	8	8	7	7	8	8	8	8	8
scout-api	90	88	90	89	87	87	88	88	87	88	87
Average (original)	48	47	49	49	48	48	48	48	47	48	48
Avg Std Dev (original)	2	3	2	2	2	3	2	3	3	2	3
Median (original)	32	33	33	33	33	32	34	33	32	33	34
95% CI-LB (original)	46	45	47	47	46	46	46	46	45	46	46
95% CI-UB (original)	49	49	50	50	50	49	49	50	49	49	50
cwa-verification	8	9	8	8	8	8	8	8	8	8	9
genome-nexus	20	19	18	19	19	19	21	20	18	19	20
gestaohospital-rest	22	22	22	22	22	22	22	22	22	22	22
ind0	42	41	43	44	42	43	44	45	39	41	47
languagetool	5	5	8	5	4	7	4	4	4	4	4
market	18	20	20	19	20	20	20	19	20	20	19
ocvn-rest	528	533	533	535	530	529	533	528	533	534	536
rest-ncs	6	6	6	6	6	6	6	6	6	6	6
rest-scs	7	8	9	10	8	7	6	9	7	8	10
restcountries	2	2	2	2	2	2	2	2	2	2	2
Average (others)	66	66	67	67	66	66	67	66	66	66	68
Avg Std Dev (others)	2	3	2	2	2	3	2	2	2	2	2
Median (others)	13	14	14	14	14	14	14	14	13	14	14
95% CI-LB (others)	64	64	65	65	64	65	65	65	65	65	66
95% CI-UB (others)	68	68	68	69	68	68	68	68	67	68	69
Average (all)	60	60	61	61	60	60	60	60	60	60	61
Avg Std Dev (all)	2	3	2	2	2	3	2	2	2	2	2
Median (all)	20	20	20	19	20	20	21	20	20	20	20
95% CI-LB (all)	58	58	59	59	58	59	59	59	58	59	59
95% CI-UB (all)	61	62	62	62	62	62	62	62	61	62	63
Max Count	5	6	8	7	6	5	7	4	5	5	10

Table 31 Line Coverage% performance for probOfRandomSampling

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	47	48	49	50	50	50	50	50	51	51	52
features-service	79	81	81	81	82	81	81	81	81	81	82
proxyprint	50	52	54	51	52	53	53	54	55	55	55
rest-ncs	91	92	92	92	93	91	92	93	92	93	90
rest-news	59	64	64	64	64	64	64	65	65	64	64
rest-scs	77	80	76	78	76	75	79	78	80	80	77
scout-api	53	52	52	52	52	52	52	52	53	52	53
Average (original)	65	67	67	67	67	67	67	68	68	68	68
Avg Std Dev (original)	4	2	2	3	2	3	3	3	3	2	4
Median (original)	59	64	64	64	64	64	64	65	65	64	64
95% CI-LB (original)	62	65	65	65	65	65	65	66	66	66	65
95% CI-UB (original)	68	69	69	69	69	68	69	70	70	70	70
cwa-verification	46	47	47	47	47	47	47	47	47	46	47
genome-nexus	34	34	35	34	34	35	34	35	34	34	34
gestaohospital-rest	39	40	39	40	39	40	39	40	39	39	39
ind0	16	16	15	15	14	15	14	13	14	13	12
languagetool	35	37	40	34	34	40	37	33	34	34	39
market	47	49	49	48	48	47	48	49	48	48	47
ocvn-rest	35	35	35	35	35	35	35	35	35	35	35
restcountries	67	71	71	70	71	72	70	70	71	71	70
Average (others)	40	41	41	40	40	41	40	40	40	40	40
Avg Std Dev (others)	2	2	2	2	2	2	2	2	2	2	2
Median (others)	37	38	40	38	37	40	38	38	37	37	39
95% CI-LB (others)	38	40	40	39	39	40	39	39	39	39	39
95% CI-UB (others)	41	42	43	42	41	43	42	42	42	41	42
Average (all)	52	53	53	53	53	53	53	53	53	53	53
Avg Std Dev (all)	3	2	2	2	2	3	2	2	2	2	3
Median (all)	47	49	49	50	50	50	50	50	51	51	52
95% CI-LB (all)	50	52	52	51	51	52	51	51	52	52	51
95% CI-UB (all)	54	55	55	54	54	55	55	55	55	55	55
Max Count	3	6	5	3	4	6	2	7	6	4	6

Table 32 Branch Coverage% performance for probOfRandomSampling

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	18	18	18	19	19	19	19	19	19	20	20
features-service	25	25	25	25	24	24	24	24	24	24	24
proxyprint	10	12	14	11	12	13	12	14	14	14	14
rest-ncs	64	66	66	66	67	65	66	67	66	66	64
rest-news	20	23	23	23	23	24	24	23	24	24	24
rest-scs	40	44	42	43	41	41	44	45	45	46	44
scout-api	23	23	22	23	22	23	22	23	23	22	23
Average (original)	29	30	30	30	30	30	30	31	31	31	30
Avg Std Dev (original)	3	2	2	2	2	2	2	2	2	2	3
Median (original)	23	23	23	23	23	24	24	23	24	24	24
95% CI-LB (original)	26	28	28	28	28	28	29	29	29	29	28
95% CI-UB (original)	31	32	32	32	32	31	32	32	32	32	33
cwa-verification	9	9	9	9	9	9	9	9	9	9	9
genome-nexus	18	18	18	18	17	18	18	18	18	17	18
gestaohospital-rest	3	3	3	3	3	3	3	3	3	3	3
ind0	3	3	3	3	2	2	2	2	2	2	2
languagetool	14	14	16	13	13	16	15	13	13	13	15
market	3	4	4	4	3	3	3	4	4	3	3
ocvn-rest	6	6	6	6	6	6	6	6	6	6	6
restcountries	50	56	56	55	56	56	56	56	56	56	55
Average (others)	13	14	14	14	14	14	14	14	14	14	14
Avg Std Dev (others)	1	1	1	1	1	1	1	1	1	1	1
Median (others)	8	8	8	8	8	8	8	8	8	8	8
95% CI-LB (others)	12	13	14	13	13	14	13	13	13	13	13
95% CI-UB (others)	14	15	15	15	14	15	15	15	14	14	15
Average (all)	20	22	22	21	21	21	22	22	22	22	22
Avg Std Dev (all)	2	1	2	2	2	2	2	2	1	1	2
Median (all)	18	18	18	18	17	18	18	18	18	17	18
95% CI-LB (all)	19	20	21	20	20	21	20	21	21	21	20
95% CI-UB (all)	22	23	23	23	22	22	23	23	23	23	23
Max Count	7	9	10	8	5	8	6	9	9	8	8

Table 33 Faults# performance for probOfRandomSampling

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	25	25	26	26	26	27	27	26	27	27	26
features-service	32	33	33	33	33	32	33	33	33	33	34
proxyprint	85	83	87	82	83	85	85	82	85	85	84
rest-ncs	6	6	6	6	6	6	6	6	6	6	6
rest-news	7	8	8	8	8	7	7	8	8	8	8
rest-scs	8	8	6	7	8	7	9	8	8	9	8
scout-api	87	90	87	87	86	87	87	89	91	88	90
Average (original)	36	36	36	36	36	36	36	36	37	37	37
Avg Std Dev (original)	2	3	2	2	2	2	2	2	3	2	2
Median (original)	25	25	26	26	26	27	27	26	27	27	26
95% CI-LB (original)	34	34	35	34	34	35	35	35	35	35	35
95% CI-UB (original)	37	38	38	37	37	37	38	37	39	38	38
cwa-verification	8	8	8	8	8	8	8	8	8	8	8
genome-nexus	20	19	20	21	20	19	20	19	19	19	19
gestaohospital-rest	22	22	22	22	22	22	22	22	22	22	22
ind0	37	42	44	43	44	43	40	43	42	40	38
languagetool	4	4	6	4	4	7	4	4	4	4	5
market	19	19	19	19	20	20	19	19	19	20	20
ocvn-rest	467	491	503	509	523	529	549	556	566	578	588
restcountries	2	2	2	2	2	2	2	2	2	2	2
Average (others)	72	76	78	78	80	81	83	84	85	87	88
Avg Std Dev (others)	3	3	3	2	2	3	2	2	3	3	3
Median (others)	20	19	20	20	20	20	20	19	19	20	20
95% CI-LB (others)	70	74	76	77	79	80	81	83	83	84	86
95% CI-UB (others)	74	78	80	80	82	83	85	86	87	89	90
Average (all)	55	57	58	58	60	60	61	62	63	63	64
Avg Std Dev (all)	2	3	2	2	2	3	2	2	3	3	2
Median (all)	20	19	20	21	20	20	20	19	19	20	20
95% CI-LB (all)	53	55	57	57	58	59	60	60	61	61	62
95% CI-UB (all)	57	60	60	60	61	62	63	63	65	65	66
Max Count	4	5	7	6	7	7	6	5	7	8	8

Table 34 Line Coverage% performance for focusedSearchActivationTime

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	47	47	50	49	49	50	49	50	49	51	51
features-service	79	81	81	81	81	81	81	81	81	81	82
proxyprint	50	50	53	52	53	53	53	54	54	54	55
rest-ncs	92	92	92	91	93	91	92	93	93	93	93
rest-news	51	63	64	64	64	64	64	65	65	64	64
rest-scs	76	77	74	78	77	75	77	76	75	78	74
scout-api	53	52	53	52	52	52	53	54	53	53	54
Average (original)	64	66	67	67	67	67	67	68	67	68	68
Avg Std Dev (original)	5	3	3	3	2	3	3	3	3	3	3
Median (original)	53	63	64	64	64	64	64	65	65	64	64
95% CI-LB (original)	60	64	65	64	65	65	65	65	65	66	66
95% CI-UB (original)	68	68	69	69	69	68	69	69	69	70	69
cwa-verification	39	47	47	47	47	47	47	47	47	47	47
genome-nexus	30	33	34	34	34	35	34	35	35	34	35
gestaohospital-rest	39	39	39	39	40	40	39	40	40	40	40
ind0	10	12	12	15	13	15	15	16	17	17	16
languagetool	20	34	40	37	39	40	37	36	36	35	36
market	48	48	48	48	48	47	47	48	48	48	48
ocvn-rest	34	35	35	35	35	35	35	35	35	35	35
restcountries	68	70	70	71	71	72	70	70	71	71	72
Average (others)	36	40	41	41	41	41	40	41	41	41	41
Avg Std Dev (others)	3	3	2	2	2	2	2	2	2	2	2
Median (others)	36	37	40	38	40	40	38	38	38	38	38
95% CI-LB (others)	34	38	39	39	40	40	39	40	39	39	39
95% CI-UB (others)	38	42	42	42	42	43	42	42	43	43	43
Average (all)	49	52	53	53	53	53	53	53	53	53	53
Avg Std Dev (all)	4	3	2	3	2	3	2	2	2	3	2
Median (all)	48	48	50	49	49	50	49	50	49	51	51
95% CI-LB (all)	46	50	51	51	52	52	51	52	51	51	52
95% CI-UB (all)	52	54	54	55	55	55	55	55	55	55	55
Max Count	1	3	4	4	5	6	2	8	8	8	11

Table 35 Branch Coverage% performance for focusedSearchActivationTime

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	17	18	19	18	19	19	19	20	19	20	20
features-service	24	24	23	24	24	24	25	25	26	25	25
proxyprint	11	10	13	13	13	13	12	13	14	14	14
rest-ncs	65	66	65	64	66	65	66	66	66	67	67
rest-news	16	22	23	23	24	24	24	24	23	23	24
rest-scs	44	42	40	44	43	41	44	41	41	44	40
scout-api	23	22	23	23	23	23	22	24	23	23	23
Average (original)	29	29	29	30	30	30	30	30	30	31	30
Avg Std Dev (original)	3	2	2	3	2	2	2	2	2	2	2
Median (original)	23	22	23	23	24	24	24	24	23	23	24
95% CI-LB (original)	26	28	28	28	29	28	29	29	29	29	29
95% CI-UB (original)	31	31	31	32	32	31	32	32	32	33	32
cwa-verification	8	9	9	9	9	9	9	9	9	9	9
genome-nexus	13	17	17	17	17	18	18	18	18	17	18
gestaohospital-rest	3	3	3	3	3	3	3	3	3	3	3
ind0	0	2	2	2	2	2	3	3	4	3	3
languagetool	7	13	16	15	15	16	15	14	14	14	14
market	4	4	4	3	3	3	3	4	3	4	4
ocvn-rest	6	6	6	6	6	6	6	6	6	6	6
restcountries	53	56	56	56	56	56	56	56	56	56	56
Average (others)	12	14	14	14	14	14	14	14	14	14	14
Avg Std Dev (others)	1	1	1	1	1	1	1	1	1	1	1
Median (others)	6	8	8	8	8	8	8	8	8	8	8
95% CI-LB (others)	11	13	13	13	13	14	13	13	13	13	13
95% CI-UB (others)	13	15	15	15	15	15	15	15	15	15	15
Average (all)	20	21	21	21	22	21	22	22	22	22	22
Avg Std Dev (all)	2	2	2	2	1	2	2	2	2	2	1
Median (all)	13	17	17	17	17	18	18	18	18	17	18
95% CI-LB (all)	18	20	20	20	21	21	20	21	21	21	21
95% CI-UB (all)	21	22	22	23	23	22	23	23	23	23	23
Max Count	4	5	6	5	5	7	7	9	8	9	10

Table 36 Faults# performance for focusedSearchActivationTime

SUT	0	0.1	0.2	0.3	0.4	<u>0.5</u>	0.6	0.7	0.8	0.9	1
catwatch	24	25	25	26	27	27	25	27	26	27	27
features-service	31	32	33	33	32	32	33	33	33	33	33
proxyprint	83	82	83	83	82	85	86	85	84	87	85
rest-ncs	6	6	6	6	6	6	6	6	6	6	6
rest-news	7	7	8	8	8	7	8	8	8	7	8
rest-scs	10	8	7	7	8	7	8	7	8	8	7
scout-api	87	87	88	87	88	87	88	92	89	93	91
Average (original)	35	35	36	36	36	36	36	37	36	37	37
Avg Std Dev (original)	2	3	2	2	3	2	2	2	2	3	2
Median (original)	24	25	25	26	27	27	25	27	26	27	27
95% CI-LB (original)	34	33	34	34	34	35	35	35	35	35	35
95% CI-UB (original)	37	37	37	37	38	37	38	38	38	39	38
cwa-verification	4	8	8	8	8	8	8	8	8	8	8
genome-nexus	21	20	20	20	20	19	20	20	18	19	20
gestaohospital-rest	22	22	22	22	22	22	22	22	22	22	22
ind0	18	32	36	41	41	43	43	46	50	44	44
languagetool	4	4	7	5	4	7	4	4	4	5	4
market	19	19	19	19	19	20	20	19	19	20	19
ocvn-rest	442	467	483	499	520	529	542	548	561	570	577
restcountries	2	2	2	2	2	2	2	2	2	2	2
Average (others)	66	72	75	77	80	81	83	84	86	86	87
Avg Std Dev (others)	3	2	3	2	2	3	2	2	2	3	3
Median (others)	18	20	20	20	20	20	20	20	18	20	20
95% CI-LB (others)	64	70	72	75	78	80	81	82	84	84	85
95% CI-UB (others)	69	73	77	79	81	83	84	85	87	88	89
Average (all)	52	55	56	58	59	60	61	62	63	63	64
Avg Std Dev (all)	3	2	3	2	2	3	2	2	2	3	2
Median (all)	19	20	20	20	20	20	20	20	19	20	20
95% CI-LB (all)	50	53	54	56	57	59	59	60	61	61	62
95% CI-UB (all)	54	57	58	59	61	62	63	63	64	65	65
Max Count	5	4	7	6	6	7	7	7	7	9	8

Table 37 Line Coverage% performance for baseTaintAnalysisProbability

SUT	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	<u>0.9</u>	1
catwatch	50	50	49	49	49	48	48	48	49	50	48
features-service	81	81									
languagetool	23	36	39	33	35	34	38	38	37	40	35
proxyprint	52	54	54	53	53	54	53	53	52	53	53
rest-ncs	92	92	92	93	93	93	92	92	91	92	
rest-news	64	64	65	64	65	64	64	65	65	64	65
rest-scs	78	78	80	78	79	76	75	79	77	75	76
restcountries	69	70	73	70	70	70	70	70	70	72	71
scout-api	53	52	52	52	53	52	52	53	53	52	52
Average (original)	62	64	65	64	64	64	64	64	64	64	64
Avg Std Dev (original)	3	3	4	3	3	2	3	3	3	3	3
Median (original)	64	64	65	64	65	64	64	65	65	64	65
95% CI-LB (original)	60	62	62	62	62	62	62	62	62	62	62
95% CI-UB (original)	65	66	68	66	66	65	66	66	66	66	66
cwa-verification	39	47	47								
genome-nexus	33	35	33	33	34	34	34	36	34	35	36
gestaohospital-rest	40	40									
ind0	11	12	12	14	14	13	15	16	16	15	15
market	48	47	48	49	47	48	48	47	47	47	48
ocvn-rest	35	35	36	35	35	35	35	35	35	35	35
Average (others)	34	36	36	36	36	36	36	37	36	36	37
Avg Std Dev (others)	1	1	2	1	1	2	2	1	2	2	1
Median (others)	37	38	38								
95% CI-LB (others)	33	35	34	35	35	35	35	36	35	36	36
95% CI-UB (others)	35	37	37	37	37	37	38	38	38	37	37
Average (all)	51	53	53								
Avg Std Dev (all)	2	2	3	2	2	2	2	2	2	3	2
Median (all)	50	50	49	49	49	48	48	48	49	50	48
95% CI-LB (all)	49	51	51	51	51	51	51	52	51	52	51
95% CI-UB (all)	53	54	55	54	55	54	55	55	55	55	54
Max Count	4	5	8	5	6	5	3	7	6	5	5

Table 38 Branch Coverage% performance for baseTaintAnalysisProbability

SUT	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
catwatch	19	18	19	19	19						
features-service	24	24	24	25	25	24	24	24	25	24	25
languagetool	10	14	15	13	14	13	14	15	15	16	14
proxyprint	11	14	13	14	13	14	13	13	13	13	13
rest-ncs	66	66	66	66	67	67	66	67	66	65	67
rest-news	23	24	24	24	24	23	23	24	23	24	24
rest-scs	46	45	47	44	44	42	42	45	42	41	42
restcountries	54	55	56	56	56	55	55	55	56	56	56
scout-api	23	22	22	23							
Average (original)	31	31	32	32	32	31	31	32	31	31	31
Avg Std Dev (original)	2	2	2	2	2	2	2	2	2	2	2
Median (original)	23	24	24	24	24	23	23	24	23	24	24
95% CI-LB (original)	29	30	30	30	30	30	30	30	30	30	30
95% CI-UB (original)	32	33	33	33	33	33	32	33	33	32	33
cwa-verification	8	9									
genome-nexus	16	18	16	17	17	18	18	19	17	18	19
gestaohospital-rest	3										
ind0	1	1	2	2	2	2	2	3	2	2	2
market	3	3	4	4	3	3	4	3	3	3	3
ocvn-rest	6										
Average (others)	6	7									
Avg Std Dev (others)	1	1	1	1	1	1	1	1	1	1	1
Median (others)	4	4	5	5	4	4	5	4	4	4	4
95% CI-LB (others)	6	6	6	6	6	6	6	7	6	7	7
95% CI-UB (others)	7	7	7	7	7	7	7	8	7	7	7
Average (all)	21	22	22	22	22	21	21	22	21	21	22
Avg Std Dev (all)	2	2	2	2	2	1	2	2	2	2	1
Median (all)	16	18	16	17	17	18	18	18	17	18	19
95% CI-LB (all)	20	20	20	20	21	20	20	21	20	21	20
95% CI-UB (all)	22	23	23	23	23	22	22	23	23	22	23
Max Count	4	6	8	10	9	7	6	8	7	8	10

Table 39 Faults/# performance for baseTaintAnalysisProbability

SUT	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	<u>0.9</u>	1
catwatch	28	26	26	26	26	26	26	26	26	27	25
features-service	33	34	33	34	34	33	33	33	34	32	32
languagetool	4	4	4	4	4	4	4	5	4	7	4
proxyprint	85	84	83	81	84	82	84	81	83	85	85
rest-ncs	6	6	6	6	6	6	6	6	6	6	6
rest-news	8	8	8	8	8	8	7	8	8	7	8
rest-scs	11	11	11	10	10	10	9	9	8	7	7
restcountries	2	2	2	2	2	2	2	2	2	2	2
scout-api	89	88	84	87	91	87	91	92	91	87	87
Average (original)	30	29	29	29	29	29	29	29	29	29	28
Avg Std Dev (original)	1	1	2	2	2	1	2	2	2	2	1
Median (original)	11	11	11	10	10	10	9	9	8	7	8
95% CI-LB (original)	29	28	27	27	28	28	28	28	28	28	27
95% CI-UB (original)	31	30	30	30	31	30	30	30	30	30	30
cwa-verification	4	8	8	8	8	8	8	8	8	8	8
genome-nexus	20	20	20	20	20	20	20	20	20	19	20
gestaohospital-rest	22	22	22	22	22	22	22	22	22	22	22
ind0	40	43	42	43	44	43	44	44	43	43	38
market	20	20	19	20	20	20	20	20	19	20	19
ocvn-rest	532	530	533	531	535	529	531	530	532	529	532
Average (others)	106	107	107	107	108	107	108	107	107	107	106
Avg Std Dev (others)	3	3	3	3	2	2	3	3	3	3	3
Median (others)	21	21	21	21	21	21	21	21	21	21	21
95% CI-LB (others)	105	105	105	105	107	105	105	106	105	105	104
95% CI-UB (others)	108	109	110	109	110	109	109	109	109	109	109
Average (all)	60	60	60	60	61	60	60	60	60	60	60
Avg Std Dev (all)	2	2	2	2	2	2	2	2	2	3	2
Median (all)	20	20	20	20	20	20	20	20	20	20	20
95% CI-LB (all)	59	59	58	59	59	59	59	59	59	59	58
95% CI-UB (all)	62	62	62	62	63	61	62	62	62	62	61
Max Count	9	9	7	8	10	7	7	9	7	7	7

Appendix B: Comparison Tables for the Second Set of Experiments

Table 40 Vargha-Delaney A-test scores of Line Coverage for 5 parameters

SUT	A	B	C	D	E
catwatch	0.53	0.65	0.41	0.56	0.43
cwa-verification	0.53	0.53	0.65	0.55	0.49
features-service	0.37	0.53	0.45	0.51	0.68
genome-nexus	0.58	0.21	0.58	0.46	0.25
gestaohospital-rest	0.48	0.34	0.55	0.5	0.48
ind0	0.67	0.36	0.42	0.48	0.23
languagetool	0.37	0.53	0.38	0.44	0.38
market	0.64	0.62	0.7	0.6	0.56
ocvn-rest	0.55	0.6	0.5	0.5	0.65
proxyprint	0.5	0.5	0.34	0.48	0.57
rest-ncs	0.67	0.65	0.56	0.51	0.66
rest-news	0.44	0.36	0.47	0.37	0.63
rest-scs	0.68	0.74	0.44	0.63	0.7
restcountries	0.36	0.44	0.42	0.51	0.52
scout-api	0.61	0.53	0.5	0.68	0.42
Average	0.53	0.51	0.49	0.52	0.51
Median	0.53	0.53	0.47	0.51	0.52

Statistically significant values (those with p-value < 0.05) are shown in bold if $\hat{A}_{xy} \geq 0.5$ and italic if $\hat{A}_{xy} < 0.5$.

A = \hat{A}_{xy} Compare x with y for `maxSqlInitActionsPerMissingData`, where x = 1 and y = 5.

B = \hat{A}_{xy} Compare x with y for `probOfApplySQLActionToCreateResources`, where x = 0.2 and y = 0.5.

C = \hat{A}_{xy} Compare x with y for `probOfRandomSampling`, where x = 0.7 and y = 0.8.

D = \hat{A}_{xy} Compare x with y for `focusedSearchActivationTime`, where x = 0.9 and y = 0.8.

E = \hat{A}_{xy} Compare x with y for `baseTaintAnalysisProbability`, where x = 0.2 and y = 0.9

Table 41 Vargha-Delaney A-test scores of **Branch Coverage** for 5 parameters

SUT	A	B	C	D	E
catwatch	0.58	0.7	0.33	0.7	0.54
cwa-verification	0.5	0.5	0.5	0.5	0.5
features-service	0.75	0.46	0.54	0.48	0.57
genome-nexus	0.59	<i>0.19</i>	0.58	0.47	0.23
gestaohospital-rest	0.5	0.5	0.5	0.5	0.5
ind0	0.69	0.48	0.43	0.45	0.28
languagetool	0.35	0.52	0.44	0.48	0.39
market	0.61	0.65	0.62	0.62	0.62
ocvn-rest	0.5	0.55	0.5	0.5	0.55
proxyprint	0.45	0.5	0.41	0.42	0.5
rest-ncs	0.63	0.66	0.68	0.64	0.59
rest-news	0.3	<i>0.16</i>	0.47	0.5	0.56
rest-scs	0.68	0.78	0.49	0.68	0.77
restcountries	0.44	0.49	0.38	0.43	0.53
scout-api	0.56	0.44	0.44	0.51	0.21
Average	0.54	0.51	0.49	0.53	0.49
Median	0.56	0.5	0.49	0.5	0.53

Statistically significant values (those with p-value < 0.05) are shown in bold if $\hat{A}_{xy} \geq 0.5$ and italic if $\hat{A}_{xy} < 0.5$.

A = \hat{A}_{xy} Compare x with y for `maxSqlInitActionsPerMissingData`, where x = 1 and y = 5.

B = \hat{A}_{xy} Compare x with y for `probOfApplySQLActionToCreateResources`, where x = 0.2 and y = 0.5.

C = \hat{A}_{xy} Compare x with y for `probOfRandomSampling`, where x = 0.7 and y = 0.8.

D = \hat{A}_{xy} Compare x with y for `focusedSearchActivationTime`, where x = 0.9 and y = 0.8.

E = \hat{A}_{xy} Compare x with y for `baseTaintAnalysisProbability`, where x = 0.2 and y = 0.9

Table 42 Vargha-Delaney A-test scores of **Faults** for 5 parameters

SUT	A	B	C	D	E
catwatch	0.61	0.56	0.28	0.62	0.35
cwa-verification	<i>0.24</i>	0.39	0.45	0.35	0.32
features-service	0.43	0.67	0.51	0.64	0.71
genome-nexus	0.63	0.32	0.44	0.58	0.5
gestaohospital-rest	0.5	0.5	0.5	0.5	0.5
ind0	0.52	0.43	0.5	0.33	0.44
languagetool	<i>0.27</i>	0.52	0.5	0.66	0.37
market	0.31	0.67	0.56	0.67	0.38
ocvn-rest	0.63	0.68	<i>0.18</i>	0.84	0.63
proxyprint	0.52	0.5	0.33	0.64	0.42
rest-ncs	0.5	0.5	0.5	0.5	0.5
rest-news	0.62	0.62	0.6	0.3	0.72
rest-scs	0.62	0.62	0.48	0.5	0.86
restcountries	0.5	0.5	0.5	0.5	0.5
scout-api	0.66	0.6	0.46	0.7	0.23
Average	0.5	0.54	0.45	0.56	0.5
Median	0.52	0.52	0.5	0.58	0.5

Statistically significant values (those with p-value < 0.05) are shown in bold if $\hat{A}_{xy} \geq 0.5$ and italic if $\hat{A}_{xy} < 0.5$.

A = \hat{A}_{xy} Compare x with y for `maxSqlInitActionsPerMissingData`, where x = 1 and y = 5.

B = \hat{A}_{xy} Compare x with y for `probOfApplySQLActionToCreateResources`, where x = 0.2 and y = 0.5.

C = \hat{A}_{xy} Compare x with y for `probOfRandomSampling`, where x = 0.7 and y = 0.8.

D = \hat{A}_{xy} Compare x with y for `focusedSearchActivationTime`, where x = 0.9 and y = 0.8.

E = \hat{A}_{xy} Compare x with y for `baseTaintAnalysisProbability`, where x = 0.2 and y = 0.9

Appendix C: Time-Based Graphs for Branch Coverage from the Fourth Set of Experiments

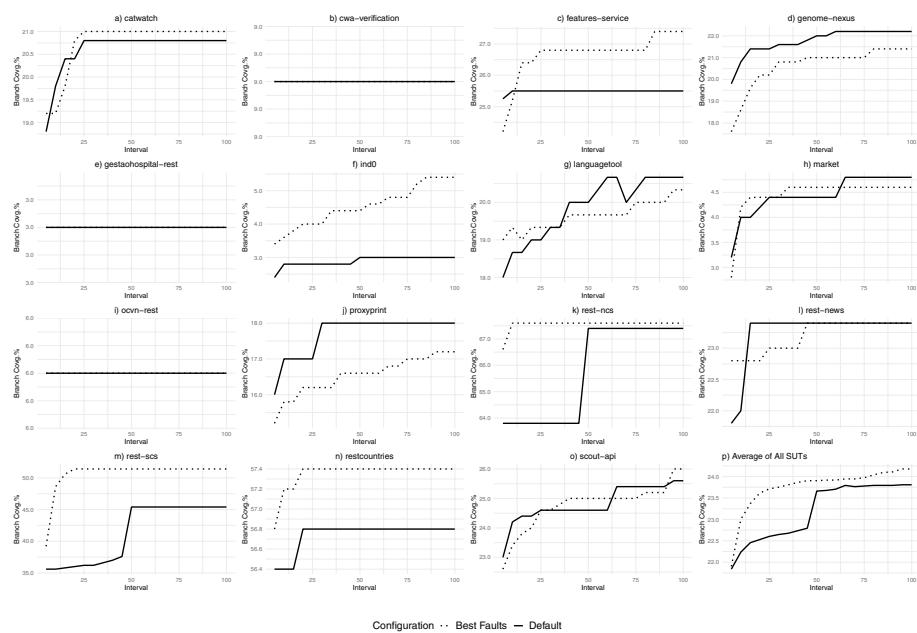


Fig. 2 Achieved branch coverage of the 15 SUTs over time based on the fourth set of experiments and their average

Appendix D: Time-Based Graphs for Faults from the Fourth Set of Experiments

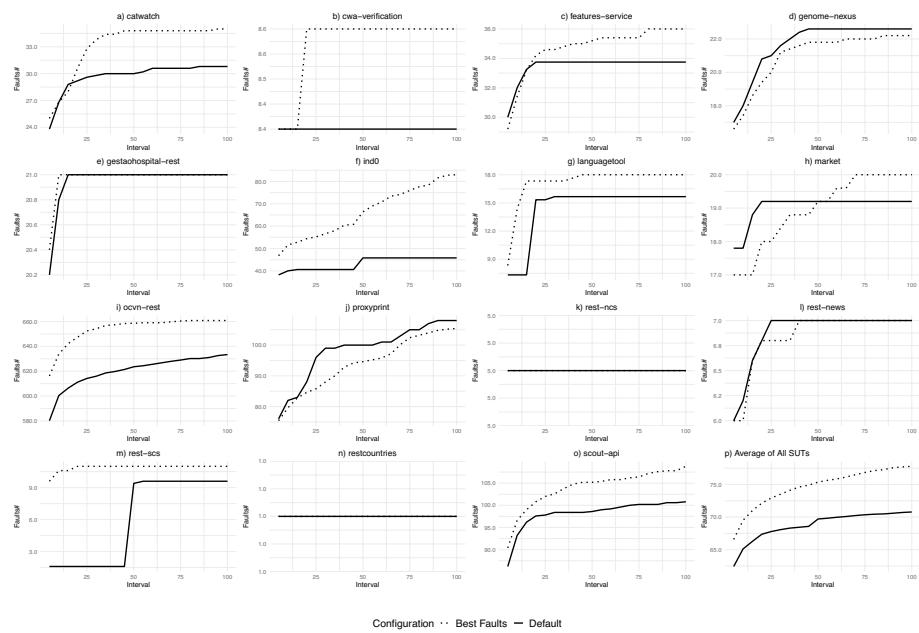


Fig. 3 Detected faults of the 15 SUTs over time based on the fourth set of experiments and their average

Acknowledgements This work is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EAST project, grant agreement No. 864972). Man Zhang is supported by State Key Laboratory of Complex & Critical Software Environment (SKLCCSE, grant No. CCSE-2024ZX-01).

Author Contributions All authors contributed to experiment design and implementation of EvoMASTER. Amid Golmohammadi conducted experiments and wrote script for analyzing results. The first draft of the manuscript was written by Amid Golmohammadi, then Man Zhang and Andrea Arcuri updated and commented on previous versions of the manuscript. All authors reviewed the manuscript and approved the final version.

Funding This work is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EAST project, grant agreement No. 864972). Man Zhang is supported by State Key Laboratory of Complex & Critical Software Environment (SKLCCSE, grant No. CCSE-2024ZX-01).

Data Availability Statement Our tool and open-source case studies employed in our experiments are publicly available online, i.e., www.evomaster.org for EvoMASTER, and <https://github.com/WebFuzzing/EMB> for open-source case studies. Due to confidentiality, we are unable to share the industry API.

Declarations

Ethical Approval Not applicable.

Informed consent Not applicable.

Conflict of Interest The authors declared that they have no conflict of interest.

Clinical trial number Not applicable.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

References

- Alshraideh M, Bottaci L (2006) Search-based software test data generation for string data using program-specific search operators. *Softw Test Verification Reliab (STVR)* 16(3):175–203
- Arcuri A (2017) Many Independent Objective (MIO) algorithm for test suite generation. In: International Symposium on Search Based Software Engineering (SSBSE), pp 3–17
- Arcuri A (2018) Test suite generation with the Many Independent Objective (MIO) algorithm. *Inf Softw Technol* 104:195–206
- Arcuri A (2019) RESTful API automated test case generation with EvoMaster. *ACM Trans Softw Eng Methology (TOSEM)* 28(1):3
- Arcuri A, Briand L (2014) A hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering. *Softw Test Verification Reliab (STVR)* 24(3):219–250
- Arcuri A, Fraser G (2013) Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empir Softw Eng (EMSE)* 18(3):594–623
- Arcuri A, Galeotti JP (2020) Handling SQL databases in automated system test generation. *ACM Trans Softw Eng Methodology (TOSEM)* 29(4):1–31
- Arcuri A, Galeotti JP (2021) Enhancing search-based testing with testability transformations for existing APIs. *ACM Trans Softw Eng Methodology (TOSEM)* 31(1):1–34
- Arcuri A, Galeotti JP, Marculescu B, Zhang M (2021) EvoMaster: a search-based system test generation tool. *J Open Source Softw* 6(57):2153
- Arcuri A, Zhang M, Golmohammadi A, Belhadi A, Galeotti JP, Marculescu B, Seran S (2023) EMB: a curated corpus of web/enterprise applications and library support for software testing research. In: 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). IEEE, pp 433–442
- Belhadi A, Zhang M, Arcuri A (2023) Random testing and evolutionary testing for fuzzing GraphQL APIs. *ACM Trans Web*
- Breiman L (2017) Classification and regression trees. Routledge
- Breiman L (2001) Random forests. *Mach Learn* 45:5–32
- Chicco D, Warrens MJ, Jurman G (2021) The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Comput Sci* 7:623
- Črepinský M, Liu S-H, Merník M (2014) Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. *Appl Soft Comput* 19:161–170

- Da Silva FQ, Suassuna M, França ACC, Grubb AM, Gouveia TB, Monteiro CV, Santos IE (2014) Replication of empirical studies in software engineering research: a systematic mapping study. *Empir Softw Eng* 19:501–557
- D'agostino RB, Belanger A, D'Agostino Jr RB (1990) A suggestion for using powerful and informative tests of normality. *Am Stat* 44(4):316–321
- developers: sklearn.ensemble.GradientBoostingRegressor — Scikit-learn Documentation (2025a). Accessed 21 Jan 2025. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>
- developers: sklearn.ensemble.RandomForestRegressor — Scikit-learn Documentation (2025b). Accessed 21 Jan 2025. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>
- Doerr B, Doerr C (2020) Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, 271–321
- Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm Evol Comput* 1(1):19–31
- EvoMaster (2025). Accessed 30 Sep 2025. <https://github.com/WebFuzzing/EvoMaster>
- EvoMaster Benchmark (EMB) (2025). Accessed 30 Sep 2025. <https://github.com/WebFuzzing/EMB>
- Fraser G, Arcuri A (2011) EvoSuite: automatic generation for object-oriented software. In: ACM symposium on the Foundations of Software Engineering (FSE), pp 416–419
- Freisleben B, Härtfelder M (1993) Optimization of genetic algorithms by genetic algorithms. In: Artificial neural nets and genetic algorithms: proceedings of the international conference in Innsbruck, Austria, 1993. Springer, pp 392–399
- Friedman JH (2001) Greedy function approximation: a gradient boosting machine. *Ann Stat*, 1189–1232
- Golmohammadi A, Zhang M, Arcuri A (2023) On the impact of tool evolution and case study size on SBSE experiments: a replicated study with EvoMaster. In: International symposium on search based software engineering. Springer, pp 108–122
- Harman M, Jones BF (2001) Search-based software engineering. *J Inf Softw Technol* 43(14):833–839
- Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: trends, techniques and applications. *ACM Comput Surv (CSUR)* 45(1):11
- Hastie T, Tibshirani R, Friedman JH, Friedman JH (2009) The elements of statistical learning: data mining, inference, and prediction. Springer
- Holland JH (1992) Genetic algorithms. *Sci Am* 267(1):66–73
- Hyndman RJ, Koehler AB (2006) Another look at measures of forecast accuracy. *Int J Forecast* 22(4):679–688
- Juristo N, Gómez OS (2012) Replication of software engineering experiments. In: Empirical software engineering and verification: international summer schools, LASER 2008–2010, Elba Island, Italy, Revised Tutorial Lectures, pp 60–88
- Kim M, Xin Q, Sinha S, Orso A (2022) Automated test generation for REST APIs: no time to rest yet. In: Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis. ISSTA 2022. Association for Computing Machinery, New York, NY, USA, pp 289–301. <https://doi.org/10.1145/3533767.3534401>
- Korel B (1990) Automated software test data generation. *IEEE Trans Software Eng* 16(8):870–879
- Kotelyanskiy A, Kapfhammer GM (2014) Parameter tuning for search-based test-data generation revisited: support for previous results. In: 2014 14th international conference on quality software. IEEE, pp 79–84
- López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: iterated racing for automatic algorithm configuration. *Oper Res Perspect* 3:43–58
- Lukasczyk S, Fraser G (2022) Pynguin: automated unit test generation for Python. In: Proceedings of the ACM/IEEE 44th international conference on software engineering: companion proceedings, pp 168–172
- Montgomery DC, Peck EA, Vining GG (2021) Introduction to linear regression analysis. John Wiley & Sons
- Nakagawa S (2004) A farewell to bonferroni: the problems of low statistical power and publication bias. *Behav Ecol* 15(6):1044–1045
- Pearson K (1901) LIII. On lines and planes of closest fit to systems of points in space. *Lond Edinb Dubl Phil Mag J Sci* 2(11):559–572
- Perneger TV (1998) What's wrong with bonferroni adjustments. *BMJ* 316:1236–1238
- Quinlan JR (1986) Induction of decision trees. *Mach Learn* 1:81–106
- Sayyad AS, Goseva-Popstojanova K, Menzies T, Ammar H (2013) On parameter tuning in search based software engineering: a replicated empirical study. In: 2013 3rd international workshop on replication in empirical software engineering research. IEEE, pp 84–90
- Steel RGD, Torrie JH et al (1960) Principles and procedures of statistics

- Tawosi V, Sarro F, Petrozziello A, Harman M (2021) Multi-objective software effort estimation: a replication study. *IEEE Trans Software Eng* 48(8):3185–3205
- Vargha A, Delaney HD (2000) A critique and improvement of the cl common language effect size statistics of McGraw and Wong. *J Education Behav Stat* 25(2):101–132
- Weisberg S (2005) Applied linear regression. John Wiley & Sons
- Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. *IEEE Trans Evol Comput* 1(1):67–82
- Wooldridge JM, Wadud M, Lye J (2016) Introductory econometrics: Asia pacific edition with online study tools 12 months. Cengage AU
- Zamani S, Hemmati H (2020) A cost-effective approach for hyper-parameter tuning in search-based test case generation. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp 418–429
- Zhang M, Arcuri A (2023) Open problems in fuzzing RESTful APIs: a comparison of tools. *ACM Trans Softw Eng Methodology (TOSEM)*. <https://doi.org/10.1145/3597205>
- Zhang M, Marculescu B, Arcuri A (2021) Resource and dependency based test case generation for RESTful Web services. *Empir Softw Eng* 26(4):1–61
- Zhang M, Arcuri A, Li Y, Liu Y, Xue K (2023) White-box fuzzing RPC-based APIs with EvoMaster: an industrial case study. *ACM Trans Softw Eng Methodology* 32(5):1–38
- Zhang M, Arcuri A (2021a) Enhancing resource-based test case generation for RESTful APIs with SQL handling. In: International symposium on search based software engineering. Springer, pp 103–117
- Zhang M, Arcuri A (2021b) Adaptive hypermutation for search-based system test generation: a study on REST APIs with EvoMaster. *ACM Trans Softw Eng Methodology (TOSEM)* 31(1)
- Zhang M, Marculescu B, Arcuri A (2019) Resource-based test case generation for RESTful web services. In: Proceedings of the genetic and evolutionary computation conference, pp 1426–1434

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Amid Golmohammadi¹ · Man Zhang²  · Andrea Arcuri^{1,3}

 Man Zhang
manzhang@buaa.edu.cn

Amid Golmohammadi
amid.golmohammadi@kristiania.no

Andrea Arcuri
andrea.arcuri@kristiania.no

¹ Kristiania University of Applied Sciences, Kirkegata 24-26, 0153 Oslo, Norway

² Beihang University, 37 Xueyuan Road, Haidian District, 100191 Beijing, P.R. China

³ Oslo Metropolitan University, Pilestredet 46, 0167 Oslo, Norway