Summary of Changes with respect to Thesis “Enhancing Fuzzing Efficacy: an In-Depth Exploration and Development of Fuzzing Strategies”

We thank the valuable and insightful comments from the examiners. In this revision, we attempt to address all the examiners’ concerns as follows. Note that all involved changes in the revised thesis are marked in blue.

Responses to each examiner’s comments and concerns

Examiner A

Comment $A.1$ I believe that the thesis, which heavily builds on Havoc on its first parts, needs to clarify better in the early parts of the thesis what Havoc is and what are its origins. For instance, the first occurrence of Havoc in the thesis (abstract)... I also found it confusing that when Havoc is first cited (in page 1 of the introduction)

Response: Thanks for pointing it out! In the revised paper, we refine the corresponding abstract by adding a citation “We first evaluate Havoc (first implemented in AFL [178])”. We further change the corresponding citation in page 1 of the introduction to the citation of the original AFL.

Comment $A.2$ Since several fuzzing approaches are being improved, I feel that the dissertation could say more about how the different fuzzing approaches compare to each other. For instance could we compare the Havoc-based, coverage-based and deep- learning based approaches proposed in this thesis against each other? This could be a nice study/conclusion to Chapters 3/4/5

Response: Thanks for giving us a chance to clarify. Indeed, conducting a broader comparison can yield more robust conclusions. Consequently, we have also examined the performance comparison between Havoc and deep-learning-based approaches (Neuzz and MTFuzz) in Section 3 (see Tables 3.3 & 3.4). Similar to our findings in Section 4, deep-learning-based approaches generally underperform compared to Havoc across most benchmarks. However, increasing computational resources allocated to random strategies significantly enhances their performance relative to the original versions. The discussion of Havoc and deep-learning-based approaches primarily focuses on mutation strategies within fuzzing. In the revised thesis, we have re-emphasized this point from the introduction to the background sections. Additionally, coverage guidance (i.e., coverage-based fitness functions) is a common fitness function shared by both mutation strategies. This is highlighted in the revised chapters, clarifying that coverage-based, Havoc-based, and deep-learning-based approaches represent different phases of the fuzzing process and are not directly comparable.

Comment $A.3$ The future work section is quite concise. I would encourage the author to expand this section

Response: Good suggestion! In our revised version, we categorize our future work into two directions: "1) how to improve testing efficacy in more challenging fields; 2) how to adopt AI-based techniques to further utilize the computation resources for exposing more real-world vulnerabilities." Such directions are organized to two different sections for detailed illustration.

Examiner B

Comment $B.1$ The candidate may consider slightly revising the title of the thesis. The current title may give people an impression that the authors are conducting empirical studies on existing fuzzing strategies. In fact, in addition to empirical studies, the candidate also proposes new fuzzing strategies in the second half of the thesis.

Response: Good point! We have revised the thesis title to "Enhancing Fuzzing Efficacy: an In-Depth Exploration and Development of Fuzzing Strategies".

Comment $B.2$ The candidate combines five conference papers into one thesis. While the current way of organizing these papers is acceptable, I still suggest that the candidate provides a better research framework in the introduction section. The framework should highlight the common and fundamental problems studied in the thesis and clearly present how the five research papers have addressed these problems and the relationship between the papers.

Response: Great suggestion! Firstly, we have revised the introduction to provide a clearer and more structured research framework. The revised framework now begins with an in-depth discussion of mutation strategies and fitness functions in fuzzing. We have also included a detailed illustration of fuzzing in scenarios with complex constraints and situations where coverage guidance may not be applicable.

Additionally, we have re-drawn Figure 1.1 to better illustrate the research framework. The updated figure clarifies the logical progression from mutation strategies to improvements in fitness functions, and finally to the application of these advancements in more challenging domains. This revised framework emphasizes the iterative process of enhancing fuzzing techniques and their implications for complex problem-solving.

Furthermore, we have highlighted the key research questions associated with each of these components in the revised thesis. This includes a focused discussion on the core problems studied across the five conference papers and how each paper contributes to addressing these problems. The revised introduction now clearly presents how the papers interrelate and contribute to the overall research agenda.

Comment $B.3$ There is typically no page limit for a doctoral thesis. The candidate may consider enriching the discussion of related work and provide a more comprehensive comparison between the findings of this thesis and those reported in the literature.

Response: Good point! We have incorporated additional related work into the revised version, including a comprehensive discussion on the application of program transformation in fuzzing, e.g., "Peng et al. [122] proposed T-Fuzz, which combines symbolic execution and program transformation to explore deep execution paths of the target program...Hsu et al. [64] proposed a lightweight program transformation strategy to reduce the fuzzing overhead incurred by tracing the coverage information". Furthermore, we have provided a more detailed comparison of the work presented in this thesis with existing research, highlighting both the similarities and differences, as well as the contributions of our work to the field, e.g., "SJFuzz has provided a novel perspective on fuzzing complex systems, particularly in scenarios where coverage guidance is unavailable. It explores methods to maintain or enhance the testing efficiency for intricate systems, such as the Java Virtual Machine (JVM).".

Comment $B.4$ Recently, large language models have been used to facilitate various software testing techniques. Given that there are already quite a lot of the relevant studies, I would suggest that the authors add some discussions on LLM-based fuzzing/techniques, which represent a new direction, in proper places of this thesis.

Response: Good suggestion! In our revised paper, we have included a new section in future work to discuss LLM in fuzzing, e.g., "...we intend to adopt LLMs in the existing fuzzing framework to fuzz programs with an additional verification phase. The principle of fuzzing such programs is to generate invalid seeds which can bypass the verification phase. One fascinating idea is to integrate the LLM into the existing fuzzing framework as a single mutator. By doing so, we can generate diverse valid seeds to explore different components of the target program. Moreover, by combining existing fuzzing mutators, we have a higher chance of generating invalid seeds bypassing verification phases if we only modify limited content upon the LLM-generated seeds. If we can balance the traditional mutators and LLM-based mutators by scheduling them carefully on different seeds, the fuzzing efficiency can be improved significantly and thus more vulnerabilities of critical systems can be exposed as soon as possible...".

Examiner C

Comment $C.1$ Section 2 currently discusses three loosely related topics following the general introduction to fuzzing: Havoc, program smoothing, and the JVM. While these topics are central to the thesis, this section should present a comprehensive overview of the big picture of mutation-based fuzzing. By integrating the studied topics into a cohesive framework, you can better illustrate why you have chosen to focus on them, thereby highlighting the significance of your research. This approach will also help readers understand the broader context and importance of your work.

Response: Great suggestion! We have reorganized the "Background" section according to the fuzzing framework. The revised background section begins with a discussion of the two mutation strategies addressed in the thesis: Havoc and neural program-smoothing-based fuzzers, representing lightweight and heavyweight strategies, respectively (e.g., "...Despite the extensive range of solutions available in both academic and industrial contexts for these two strategies, there is a notable lack of fundamental studies offering guidance on selecting appropriate mutation strategies for different scenarios..."). Following this, the section explores coverage guidance within the fuzzing framework and addresses the challenges for coverage guidance (e.g., "...However, current coverage-guided fuzzing methods usually necessitate complete execution for each seed, meaning that they explore program states constrained by strict dependencies between program branches..."). Next, we introduce the discussion on coverage guidance presented in Chapter 5 of the thesis. The final part of the section provides an overview of the JIT/JVM systems targeted for fuzzing in this thesis, setting the stage for Chapters 6 and 7 (e.g., "...Consequently, the primary challenge in fuzzing foundational software systems such as JVM and JIT is how to explore program states randomly while adhering to predefined constraints...").

Comment $C.2$ The connections between Chapters 3, 4, 5, and 6 can be strengthened. At present, these chapters appear as independent research topics that seem randomly selected. Enhancing the interconnections will provide a more unified narrative. Figure 1.1 can be improved in several ways to offer a clearer and more precise overview of the research presented in this thesis. The statement ”DL-based fuzzing is not a good idea to explore complicated constraints” seems overclaim. If we consider fuzz testing as a process of traversing the search space, deep learning has the potential to act as a human expert in various aspects of the search procedure: identifying seeds, defining the search space (descendants), pruning the search tree, etc. This argument should be presented with appropriate assumptions.

Response: Good suggestion! We have revised this thesis in the introduction and background to strengthen the connection between Chapters 3,4,5. Please refer to B.2 for more details. We also have redrawn Figure 1.1 by presenting a clearer research flow and removed the corresponding overclaim about “DL-based fuzzing is not a good idea to explore complicated constraints”.

Comment $C.3$ There are minor typesetting issues that need to be addressed. The author should strive to maintain consistent font sizes across figures, code snippets, plots, and tables. Some figures (e.g., Figures 4.3 and 5.4) and tables (e.g., Table 5.1) are currently difficult to read, while the font size of Table 5.2 is larger than the main body text.

Response: Thanks for pointing this out! In our revised version, we carefully redrawn/resized the corresponding figures/tables as follows:

- Figure 1.1, Figure 2.4, Figure 4.3, Figure 5.4, Figure 6.1, and all code snippets figure.
- Table 4.4, Table 5.1 (by changing the name of variants), Table 6.3 (by changing the name of variants), Table 6.4, Table 7.3.

Comment $C.4$ The Havoc mechanism mimics the ”jump” in the search space, and when a significant portion of the test inputs in this search space are interesting, this strategy effectively adds diversity to the generated test inputs within a limited testing budget. The findings in Section 3.1.4 support this argument. Therefore, the author may consider characterizing the search space of a specific fuzzer to provide deeper insights into when and why Havoc works (or does not work) for a specific testing subject. This characterization could lead to interesting enhancements to existing fuzzers. For example, Havoc could be integrated with Automated Program Repair (APR) techniques to balance the diversity (randomness) of fuzzers and the quality of tests.

Response: Excellent point! We have conducted preliminary investigations into the operational mechanism of Havoc. Havoc's mutator types are categorized into unit and chunk mutators, and it utilizes mutator stacking to control the size of the mutators applied in each mutation step. If fuzzing is conceptualized as a search process, Havoc functions as a mechanism to control the search step size. As illustrated in Figures 3.5 and 3.6 of Section 3, the preferred search step size varies at different stages for different target programs. Since Havoc generates search steps in a uniformly random manner, it can quickly adapt to the preferences of various target programs, resulting in commendable performance.

Based on this observation, we introduced Havocmab, which employs a multi-armed bandit algorithm to anticipate the target program's preferences for search step sizes and respond more swiftly, achieving promising results. Additionally, we observed (as shown in Figure 3.9) that Havoc's performance on projects like jhead is significantly inferior to that of QSYM, which integrates a constraint solver. This is due to the many equality constraints present in jhead, which are difficult to satisfy with random strategies, leading to suboptimal performance of Havoc on such programs. We are currently working on a new study that explores how to combine Havoc with tools designed to overcome equality constraints. Preliminary findings suggest that integrating Havoc with dictionary-based methods could result in significant performance improvements.

Comment $C.5$ In my opinion, neural program smoothing is a myth. Identifying any correlations between the input space and a smaller space (e.g., a differentiable neural network), or even a feature set, may serve as a guide for fuzzers. Although I do not intend to challenge the reported data in the thesis (as any correlations would be beneficial), understanding to what extent a neural program can capture the correlations between the test input space and the program's behavior space (e.g., coverage) remains a key research question. I would appreciate any researcher providing an in-depth answer to this question.

Response: Good one! In our study, we found that the essence of neural program smoothing is not in predicting the behavior of unknown branches but rather in leveraging existing data to identify which bytes in the input influence the branches that have already been explored. For instance, in a switch statement with six cases, neural program smoothing uses seeds from three of these cases to determine which key bytes might affect the remaining cases. By mutating these bytes, it increases the likelihood of accessing the other three cases. Essentially, this method operates as a form of black-box taint analysis.

Neural program smoothing proves to be particularly effective in scenarios where traditional taint analysis is not feasible, such as chip testing, demonstrating significant potential for practical applications. We are also eager to see further research on neural program smoothing, particularly in advancing its capability to predict jumps to undiscovered structures in programs.

Comment $C.6$ Fuzzing with Phantom Programs: Using phantom programs to identify useful test inputs is a brilliant idea for mitigating the search space curse, and this could be regarded as a ”formal” approach to program smoothing. Therefore, there may be potential to explore this further. Since phantom programs are programs, I expect that aggressively flattening the control flow for some parts of the program can be useful, while it may be harmful for others if incorrect program states are created for subsequent code. There is a vast design space to trade off ”what part of the code to rewrite” and ”to what extent the code is rewritten.” I appreciate that phantom fuzzing has opened up this direction of research and encourage the author to conduct future research along this line.

Response: Thank you for your insightful comment! We appreciate your recognition of the potential of using phantom programs as a method to alleviate the search space challenge in fuzzing. We agree that there is a significant design space to explore regarding which parts of the code to rewrite and the degree of rewriting applied. This trade-off is central to optimizing the effectiveness of phantom fuzzing. We are excited about the research opportunities this approach presents and are committed to investigating these aspects further in our future work.

Comment $C.7$ Fuzzing JVMs: The motivation for JITFuzz is somewhat weak: the proposed mutators for compiler testing are also applicable to other optimized compilers. There is plenty of research on testing compilers, ranging from generation-based and synthesis-based approaches to random fuzzing. The author may also consider merging Sections 6 and 7, as both JITFuzz and SJFuzz target orthogonal problems for the same subject.

Response: Thanks for giving us a chance to clarify. To conclude, the general perspective of the JITfuzz can be summarized as follow: 1) designing mutators activating corresponding features is a good idea, 2) retaining the data dependencies or fixing the data dependencies during control-flow mutation to avoid early termination can test the deeper part of the component such as JIT, 3) mutator stacking and scheduling can facilitate the fuzzing efficiency.

Meanwhile, merging Chapters 6 and 7 might miss some perspectives presented in both side. Notably, JITfuzz is a coverage-guided fuzzer designed for testing JVM JIT only while SJFuzz is essentially a generic fuzzer for JVM as a whole, i.e., JITfuzz and SJFuzz are proposed for different purposes. Specifically, the SJFuzz presents that “However, code coverage can hardly be applied for general-purpose JVM testing techniques”. On the other hand, JITfuzz can directly collect code coverage data from JIT instead of the whole JVM. Although JITFuzz and SJFuzz address similar issues, they convey distinct perspectives to the academic community. JITFuzz emphasizes the importance of maintaining constraints including data dependencies, whereas SJFuzz focuses on strategies for handling situations where coverage guidance is ineffective. Consequently, integrating these two topics into a single chapter would be quite challenging.