**Reply to Reviewer 1**

**Comments:**

**This paper introduces Bingo-E (an extension of a tool called Bingo, presented in a paper in FSE'16). Bingo-E is a binary code clone detection tool. The input to Bingo-E is a function from a compiled program (the binary) and the output is a list of functions that are detected to be clones of that binary.**

**From a technical perspective, the paper is very sound. I did not find any noteworthy flaws with the technical presentation or in the details of how the tool works. In particular I found the inlining procedures to be an interesting "trick" for extracting the information the authors needed, and the emulation is definitely more elegant than the constraint solving used in the original FSE paper. So I do not have complaints on the technical side.**

**Overall this paper is quite good and I hope to see it accepted after a quick revision.**

Thank you very much for the valuable comments and suggestions.

We have revised the manuscript by taking care of yours and the other reviewers’ valuable comments and suggestions to enhance the paper quality. New content has been added in order to better support the claims of the paper. The revisions are marked in **blue** color font in the revised manuscript.

**My concern for this paper is that the research questions in the evaluation do not quite line up with the stated objectives in the earlier sections of the paper. The paper starts by presenting a scenario of binary code clone detection: the problem of, given a large code base, can we find clones of a function within that code base? That is the standard code clone detection problem explored by many papers. In the evaluation however, this paper studies questions about finding clones across architectures and operating systems (as per the title). That is definitely interesting and useful, but it does not quite explain the whole picture. The approaches to which Bingo-E is compared (Tracy, BinDiff) are not designed for cross-OS or cross-architecture applications (even though technically they can be used for that purpose). Those tools are designed for the classic code clone search problem.**

Thank you very much for the valuable comments and suggestions.

**This paper ought to explore how well Bingo-E performs at the typical code clone search problem in addition to the cross-OS and cross-architecture problem. Think of it this way: in theory, the cross-OS problem could be solved "perfectly" by closely studying the different compilers used and mapping the different ways that those compilers generate binaries. The code clone search problem does not have a similar theoretical answer. Plus, it is more in line with the real-world scenarios outlined in the early sections of the paper and the related work.**

**What I propose that this paper does is add a new RQ comparing Bingo-E to Tracy and/or BinDiff on a typical code clone search situation. The RQ could be answered by collecting a code repository and searching that repository using Bingo-E and the competing tool for different functions on the \*same\* compiler, OS, and architecture. That would provide an apples-to-apples comparison of Bingo-E to the competing tools. Note that Bingo-E might perform worse in that scenario, and that is OK since the paper can also point to much better performance on the cross-OS and cross-architecture problems. I don't think the other RQs need to be modified. Instead, a new RQ can just be added to give the paper a more well-rounded view of the problem.**

**Again, I think this paper is pretty strong already, and see this request as a relatively small "major" revision. It does not involve collecting too much new data or new tools. Instead, it only requires the authors to rerun the tools with a different configuration (same OS, architecture, and compiler).**

**Reply to Reviewer 2**

**Comments:**

**This paper presents an approach to conduct the code search in the binary code level. The approach presented in the paper, namely, BINGO-E, extends the early work, called BINGO, published in FSE. Case studies are conducted to compare the proposed approach, BINGO-E with the authors’ early work, BINGO, in the different configurations, such as different OS, compilers and architectures.**

**Overall, the approach seems solid and well tested in the case studies. However, I found this paper is very hard to follow. I have to read the content several times in order to grasp what the authors really means.**

Thank you very much for the valuable comments and suggestions.

We have revised the manuscript by taking care of yours and the other reviewers’ valuable comments and suggestions to enhance the paper quality. New content has been added in order to better support the claims of the paper. The revisions are marked in **blue** color font in the revised manuscript. We try to make the paper more readable, and details of the revisions can be found below.

**Below is more detailed comment:**

1. **It is not clear about the motivation of work. The authors didn’t show the usefulness of the work. In my view, the applications of such approach seem very small. Are there any adaptions in the real world applications?**
2. **It is not clear how the authors detect the similarity in the binary code. Did authors use any third party tools?**
3. **The presentation of the paper needs improvement. Concepts need better explanation. For example, selective inlining, UNICORN, 3D-CFG, QEMU and one signature binary function should be better explained to improve the readability of the paper. The authors like to use acronyms, such as BB and C1 (in section 2.3), which make the paper hard to read. Such acronyms could be avoided.**
4. **In Section 3.1, the authors mentioned that 6 types of high-level features are extracted. Are the 6 types complete?**
5. **The performance of the proposed approach is mainly compared with the authors’ previous work. This may result in biases. The approach should be compared with the approaches other than BINGO.**
6. **The design of the case study is not clear. In the code search related evaluation, it usually needs a set of queries for the code examples, benchmarks to compare with the search results, the performance measures on the ranking of the returned code examples. The paper doesn’t provide any of the details for evaluating the results. For example, do the authors create any benchmarks to compare with the result of the binary code searches? How many queries are used in the evaluation?**

**Reply to Reviewer 3**

**The paper describes a tool for searching for binary fragments semantically equivalent to a given one. The search is performed across different architectures and different OSes. The task is clearly impossible in general, thus the reported results (about 60-80% hit rates in several cases) should be taken as a success.**

**The paper is well written are the description of the heuristic techniques used is clear enough to give to the readers some confidence in their potentialities. The reported experimental activity spans over all the claims of the authors and supports them.**

Thank you very much for the valuable comments and suggestions.

We have revised the manuscript by taking care of yours and the other reviewers’ valuable comments and suggestions to enhance the paper quality. New content has been added in order to better support the claims of the paper. The revisions are marked in **blue** color font in the revised manuscript.

**The only remark I have on this work is about the reproducibility of results: as far as I understand the piece of software described in the paper is not publicly available. Even the web site in which the data collected for the FSE 2016 paper (https://sites.google.com/site/bingofse2016) has just data tables, and not the artifacts on which these data were collected. I feel this as a bit unfair, at least with respect to the authors of the other publicly available tools the authors considered and used in their research. Ironically, one of the use cases by which the authors motivate their work is finding GPL violations, i.e., a software licence invented exactly for reducing free riders of free software and impose reciprocity of sharing.**

**Reply to Reviewer 4**

**Summary:**

**This paper incorporates various features from different categories (e.g., structural features and high-level semantic features) for accuracy improvement of the existing tool BINGO. More specifically, they combines existing feature engineering techniques such as 3D-CFG and environment values via binary code emulation for the accuracy improvement. Their experimental results demonstrate that BINGO-E outperforms BINGO in terms of the accuracy and efficiency in the cross-architecture, cross-OS, cross-compiler and intra-compiler setting.**

Thank you very much for the valuable comments and suggestions.

We have revised the manuscript by taking care of yours and the other reviewers’ valuable comments and suggestions to enhance the paper quality. New content has been added in order to better support the claims of the paper. The revisions are marked in **blue** color font in the revised manuscript.

**Strength:**

1. **Detailed comparison and contrast of existing feature engineering techniques, in terms of their strength and limitations.**
2. **Clear problem statement**

**Comments:**

1. **The paper is not self-contained. The author spent too many efforts on the discussion of the previous work BINGO, instead of BINGO-E. For example, the author spent two paragraphs on average for the result discussion of BINGO in Section 7.3.1, 7.4.1, 7.5.1 and 7.6.1.**
2. **The key contribution of this paper is missing. It seems that they tackle the fundamental issues in the binary code search problem by combing existing feature extraction techniques without the proposal of any new feature engineering technique or matching algorithm. The 3D-CFG technique or environment values via binary code emulation has been used in [26] and [20]. Although the application scenario could be different, the author at least should have the detailed discussion on the new challenge when applying into binary code search scenario. Unfortunately, they fail to give such explanation.**
3. **The evaluation section can be improved. The author claims that they compare BINGO-E with state-of-art techniques, but they only give the evaluation results with the previous work BINGO. How about DiscoRE? Multi-MH? It seems that centroid approach in 3D-CFG [26] misses the graph structure information considered by DiscoRE and Multi-MH. Without the comparison, it is hard to convince BINGO-E is more accurate.**
4. **It is not clear what the purpose of selective in-lining is? Is that used to diversify the signature set of query function? However, sometimes, we do not have the source code of the binary query function. In this case, how can they conduct the selective in-lining? Besides, in the evaluation section 7.2.1, the discussion of impact of selective in-lining is confusing. What is the experiment setting for the selective in-lining experiment? If the selective in-lining is applied to unify the test dataset, it by design will improve the accuracy. This is because the control flow graphs all the binary functions in the test dataset could not be changed. It seems that the author try to improve the search accuracy by changing the test dataset instead of improving their search techniques. In practical, not all the functions will be selectively in-lined. Therefore, it seems that this experiment setting is no-sense. It cannot demonstrate any strength of BINGO-E.**