It must be the introduction of our paper is not clear enough about the use case of Lancet that all reviewers share the false impression that our goal is to generate tests that reveal performance issues. Given this, they have raised reasonable doubts about the validity of our motivation: if one already knew which loop is the bottleneck in a program why bother generate tests for it at all? It is nice to uncover performance bottlenecks with generated tests, but that is not what Lancet is meant to achieve.

The target users for Lancet are testers who need to write performance tests for the code that may not be developed by them. This is the common case in software industry where developers only write unit tests to verify functional correctness while it is the responsibility of testers to develop integration tests to evaluate the system as a whole, often without fully understanding the entire codebase. Performance test is one important type of integration test that requires a fine-granularity exposure to the entire system to develop. Testers usually either take a white-box approach by reading the code to identify performance critical parts of the system and deducing reversely the inputs that would trigger these critical parts, or take a black-box approach to search heuristically through the space of all possible inputs guided by the cost of execution.

Lancet is set out to aid the white-box approach of performance test generation, specifically the reverse deduction process where the tester has identify a performance-critical code region but needs to trace back to the entry of the program to generate a valid test that would trigger the target code region. Or it is possible for the author of the program to provide hints on the performance critical parts and the tester just needs to run Lancet to generate tests to target these code regions. Traditional coverage-optimized test generation techniques using symbolic execution are not harnessed to accomplish this task, since they would not cover a loop for more than a few iterations before jumping to other unexplored code. The loop-centric path scheduling of Lancet is the key technical difference between Lancet and prior symbolic execution based test generation tools, which enables Lancet to steer the execution path to a high number of iterations as long as the program allows.

We admit the lack of significant evaluation of Lancet against real applications and are determined to improve on this aspect of our work in the future. But the accusation of Lancet being an example of an extreme case of bait-and-switch is a bit harsh. We do not make any claim in any part of our paper that Lancet is evaluated against real distributed applications. We discussed the scalability issues in these applications at the beginning of our paper to emphasize the prevalence and importance of performance testing.

Lancet is still under active development. From the comments from the reviewers, we realized problems in our work that we did not notice before the submission and are inspired on ways to improve our tool. We want to thank the reviewers for these invaluable feedback on our work at this early stage.