Rebuttal 6217: We thank the reviewers for their feedback. Due to limited space, we couldn't address all comments. All code/experiments will be included in appendices. We use this notation for brevity: ***Reviewer 1, 2, and 3 (R1, R2, R3)***.

R1#1 - *Technical Depth and Novelty:* We agree autoencoders (AEs) used for anomaly detection (AD) is not novel. We believe the novelty of our system is in how we combine multiple techniques from systems and ML into a new state-of-the-art (SOTA) solution as shown empirically. Although widely studied, to our knowledge, the software engineering (SE) community lacks such systems+ML software performance AD systems. Compared to the two prior SOTA works we analyzed, AutoPerf demonstrates that systems+ML tools may offer a path to more effective solutions.

R1#2 - *Analyzing Code Changes/Commit Logs*: We are slightly confused by your comment. If you are referring to using commit log comments (or other comments) to guide AutoPerf, there is evidence that such data can be erroneous (39% false positive rate) [iComment, SOSP07]. If you are referring to using actual changes in code, AutoPerf does, in fact, use such information. It just performs the analysis at the function-level rather than the statement-level.

R1#3 - *Synthetic vs. Real Bugs*: The performance bugs reported in Table 1 are *all* known performance bugs (confirmed by developers) in real-world and benchmark applications. None are synthetically generated (more details in R3#4).

R1#4 - *Performance Mutations (Good vs. Bad Changes)*: AutoPerf provides performance diagnostics, via hardware performance counters, which identifies performance improvements and eliminates them from regression candidacy. These false positives are also found and eliminated by comparing wall-clock execution time during regression tests.

R1#5 - *Number of Functions Tested*: As regression tests generally only focus on modified code, AutoPerf only analyzes the functions that have been modified between two candidate versions of a program. Therefore, the total number of functions analyzed is a byproduct of the total number of modified functions for the analysis we performed. In our case, the number of modified functions ranged from 3 to 27 out of over +1000s, as you correctly observed (e.g., Boost).

R2#1 - *Examples and Executions*: The number of application executions is equal to the number of available relevant regression tests. The number of examples and example executions are, therefore, a byproduct of the test suite for each application. Performance anomalies in some applications do not occur for all test cases. Therefore, the number of anomalous runs are reduced for these applications.

R2#2 - *False Positive (FP) & Negative (FN) Rate*: In our experiments, AutoPerf produces zero FNs. We believe this behavior is critical for a production-quality software performance bug detection tool. This is because if even one FN (an overlooked performance bug) is deployed, it could have catastrophic quality of service or life-threatening consequences (e.g., real-time critical systems). However, FNs and FPs are directly correlated. In the literature, FPs tend to increase as FNs decrease and vice versa. Given that AutoPerf has a 0% FN rate, we believe a 20-30% FP rate is reasonable, is a notable improvement over prior SOTA, and is the preferred behavior FN/FP trade-off for this type of system.

R2#3 - *Considering Only Changed Functions*: While we didn't observe any performance bugs in unchanged functions caused by code changes elsewhere, we acknowledge this is a valid scenario. Yet, we believe such cases could be readily identified via data dependency, control flow, and call graph analysis into the changed functions.

R3#1 - *Comparison with Unsupervised Density Modeling Techniques*: AutoPerf's AE design has notably improved accuracy over the prior SOTA in detecting complex, real performance bugs in real-world applications. We also tested a GMM approach to identify performance regressions in MySQL and compared it against AutoPerf for Area Under ROC Curve (AUC). Our results illustrate that GMM has significantly reduced accuracy for this application. We would be happy to include and expand our GMM comparison for all of the applications in the camera-ready version of the paper.

R3#2 - *Distribution of Reconstruction Errors*: The skewness in the reconstruction error distributions of all test applications ranges from -0.46 to 0.08. The kurtosis ranges from 1.96 to 2.64.

R3#3 - *End-to-End Timing Analysis*: End-to-end timing includes time spent in regression testing, data collection, and inference of AutoPerf's AEs. Amongst these, data collection and inference time are overheads added by AutoPerf. However, by using HPCs we significantly reduce data collection computational overhead, which is shown in Figure-6(b). We found that the inference time of AutoPerf's AEs is negligible in the overall end-to-end time.

R3#4 - *Importance of Types of Tested Regressions*: We only experimented with real-world performance anomalies in multithreaded programs. We did not inject any bugs artificially. Moreover, these types of bugs can be notoriously challenging to detect and root-cause, even for expert programmers, and their negative impact can be catastrophic. For example, the false sharing performance regression in MySQL was introduced by an expert parallel programmer attempting (and failing) to fix a lock contention issue. This single performance bug introduced 6x performance reduction in MySQL for a common code path. We will add more details about all regression cases in our final version.

R3#5 - *Limitations*: AutoPerf's accuracy is proportional to the availability and distribution of regression testing data. It isn't designed to find performance bugs that can't be captured by HPCs. We'll tone down the strength of this claim.