Dear Reviewers:

We wish to thank you for the time and effort you have spent reviewing our paper.

To answer the reviewer 1’s question, we will tell more details of our first experiment on code classification below.

In section 3.1, we mentioned the following: “We specifically design the data set this way because, many methods for the same problem tend to use the same or similar set of identifiers, which is not true in real world application.” The detailed reason for this setup is as follows.

Because time is limited during the code jam contest, contesters’ choices of identifier names tend to come from the words that appear in the question. For example, in the question “counting Sheep” of Code Jam 2016, “count” and “sheep” appear in the code of many contesters, while these two words, especially “sheep” have very little probability to appear in the code of other questions. This makes the distribution of identifier names in the code differ significantly from question to question, while the difference is much smaller among the code for the same question. This characteristic gives LEA and LES some unfair advantage, since these two methods are based on the semantic similarity between words, thus they tend to classify programs with the word “sheep” together, correctly, without looking at the program structures. Therefore we chose not to train and test on the same set of questions.

Because we train and test on two disjoint set of questions, our problems becomes a “clustering problem with labels”. That is, the result will give the class labels (0-5) for each input code block, but such label can’t be immediately mapped to a question. Therefore, we chose to consider all possible label-to-question mappings (assignments) to produce a final question label so that F1 can be computed.

Reviewer 2 mentioned four weaknesses of our paper and want us to address the weaknesses. Then, our response to these four weaknesses are as follows (the sequence is the same as weaknesses):

(1)In section 1 (introduction), we mentioned that we tested one state-of-the-art template based method (McBurney and McMillan 2014) but it commonly produced irrelevant comments based on templates. The code of this template based method along with test data is available at: https://www3.nd.edu/~pmcburne/summaries/.

(2)Even though the high level programs that we are developing today are more readable, having a quick and good descriptive comments at the beginning allows one to make informed decisions about whether to go on reading the function/block or not. This surely improves programmers’ productivity.

(3)Our method is independent of the source programming language as long as we have access to the parse tree of the input program.

(4)This paper aims to generate comments from source code. The problem of generating comments for bytecode is beyond the scope of this paper but is certainly interesting. One immediate thought would be to take advantage of the data flow graphs in the bytecode. If so, similar techniques to this paper may be applied on that type of graphs.

Thanks for the three reviewers’ comments and suggestions again.