We thank the reviewers for their valuable feedback. We will address the comments and the concerns as follows.

R1-Motivation. Our motivation is to solve the binary source code matching problem, which is very important for computer security. It can be leveraged for code vulnerability analysis and malware detection. Given the binary code, the researchers want to explore its specific meaning. A general method is using a decompiler to translate the binary code into pseudo code, which requires a lot of expert experience and has low accuracy. At the same time, the final pseudo code has poor readability because of the missing variable names and the complex "goto" statements. The situation is even serious at function-level because much more information is lost, so researchers expect to find the real corresponding source code. Traditional methods use the code literals to search the source code, but with very low top1 score (40%). Our deep learning model could help capture the potential semantic features between the source/binary code and achieve 90% top 1 score. The main contribution of our paper is using deep learning models to make this problem from "cannot" to "can". Our research can benefit tens of thousands of reverse engineering researchers.

R1&R2&R4-Novelty. In our paper, the technique is not deep in the modeling part. However, we strongly agree with R1&R4 that this should not be taken as a limitation because getting amazing results with such a simple method proves the capability of deep learning on this task, which is exactly what we want to show. Based on our CodeCMR framework, we tried many useful novel methods, including Circle Loss, Cross-Batch Memory, Adversarial Loss, and Cross-lingual Language Model. We could add a discussion part to talk about these methods, but we still think they are not the focus of this paper because our goal is to use AI to solve valuable problems. Also, we propose the integer-LSTM for code literal modeling, and a simple but useful norm weighted sampling method which helps improve 10% top1 score.

R1&R3-Why Treating Source Code as Text. Treating source code as text and getting such good results has surprised many experts. In fact, it is a salient point of our paper. Before explanation, we would like to mention that "CFG" is short for "control flow graph" (a graphical representation of binary code) in our paper rather than "context free grammars", we are not sure whether this leads to the misunderstanding. The traditional approach is to parse the codes into graphical inputs. There are two non-negligible indicators in this process: time and success rate, because function-level code loses much information such as "#include" and "#define". For example, it takes about 20 seconds to parse a piece of source code into code property graph with 84.1% success rate on 10,000 pieces. These scores are unacceptable since we want to create a solid method which can be applied to our million-to-billion-level scenario rather than a toy model. Treating source code as text has no time cost and achieves 100% success rate. Surprisingly, when we compare the results of our method with parsing models using a successfully-parsed dataset, we find that our method's top1 score is only 0.5% - 1% less. Another advantage is that our method does not need much expert experience, so the AI/NLP researchers can transfer their experience on this task easily, which could help promote the development of this field.

R1&R3-Writing. (1) We totally agree with R1&R3 that it is more canonical to explain the equations more thoroughly. However, these sentences took up about a quarter of a page, and we had to abridge them because we want to keep other sentences for the readability and consistency of the paper. This problem could be solved easily because we will have an additional page on camera ready version. (2) For Table 1, some additional explaining sentences will be added on camera ready version too. The second section of Table 1 ignores the code literals, and the third section only uses code literals. Since we propose different models on code semantic feature and code literal feature, we should do ablation study to explore the effect of each part. And in the fourth section the RANDOM line means the combination of the best models on code semantic feature and code literal feature with random sampling method, which is DPCNN+HBMP+Integer-LSTM(Integer)+Hier-LSTM(String). By comparing the RANDOM line with DPCNN+HBMP, Integer-LSTM(Integer) and Hier-LSTM(String), we could observe that the code semantic feature is much more useful than the code literal feature, which meets our assumption. To illustrate the effect of value s, we conduct the experiments on s=8 and s=10, it turns out that s=5 is a decline point. We will add a figure to show this.

R1&R3&R4-Dataset. We decide to publish our dataset and introduce the details of how we build and how to use it. We first extract files from many open-source libraries including openssl, glibc, etc. Then we compile them on 32 combinations of different compilers (gcc/clang), different platforms (x86/x64/arm/arm64) and different optimizations (O0/O1/O2/O3). About 80,000 files are successfully compiled, then we split them into about 1,700,000 functions. We randomly sample 50,000 functions and split them into trainset, validset, and testset (3:1:1). The functions are saved in a .pkl file with 33 columns, in which the first column contains source code, and 2nd-33rd columns contain binary code.

R3-Details. (1) For "source to binary" task, it is feasible to compile and compute similarity scores. Our method is more powerful. Firstly, we could save much time from compiling the code under many combinations, because our method could achieve amazing results on different combinations (more details could be found in supplementary material Figure 1). Secondly, sometimes the function-level code could not be successfully parsed because of information loss, while our method could always work. (2) The source code order is not useful on high-level. For example, "if A else B" equals to "if B else A". We have tried positional encoding and LSTM to capture the order feature, but the score declines. (3) We think the answer of the boarder impact question is yes, and we will add it on camera ready version. However, we believe that our research has more positive outcomes because the majority of researchers are protectors rather than attackers.