Answers to Reviewer Questions

*A. Answer to **Review_A_Q4**: Can you comment on how GUIDER would work when there are multiple bugs in the code?*

Multiple bugs in the code can be understood as multiple faulty layers. In this case, GUIDER still calculates the suspiciousness score for each layer. Although different failing test cases may cover different neurons, this does not necessarily result in lower suspiciousness scores for the faulty layers. As long as the failing test cases activate more neurons in the faulty layers, the suspiciousness scores for the faulty layers will be higher than those for the non-faulty layers.

```
model = Sequential()
model.add(Flatten())
model.add(Dense(units=8, activation='sigmoid'))
model.add(Dense(units=16, activation='sigmoid'))
model.add(Dense(units=32, activation='sigmoid'))
model.add(Dense(units=64, activation='sigmoid'))
model.add(Dense(units=128, activation='sigmoid'))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
  optimizer='adam', metrics=['accuracy'])
```

Fig. 1. Bug #65777704 in our dataset

Figure 1 shows a code snippet from StackOverflow Post #6577704. This code snippet constructs a classification model for image classification, but it achieves low accuracy. Because the code from the original post crashes when run directly, we added a `Flatten` layer to make it run correctly. The suggestion in the post's answer advises changing the activation function from `sigmoid` to `relu`, indicating that the erroneous layers are from the second to the sixth layers. The output of GUIDER(Ochiai) is `[0.821, 0.453, 0.825, 0.821, 0.796, 0.174, 0.463]`. Therefore, after removing the `Flatten` layer we added, sorting the model layers in descending order of suspicion gives us `[3, 4, 5, 7, 2, 6]`, which shows that the third layer has the highest suspiciousness value.

> Answer to Review_A_Q4: In scenarios where a model has multiple erroneous layers, GUIDER is still capable of placing the erroneous layer in the top-1 position.

*B. Answer to **Review_B_Q1**: In the last layer of the model shown in Listing 1, does any neuron count as activated?*

This example illustrated in Listing 1 in our paper is relatively complex. We apologize for not providing a detailed description of how trained models run with test cases. Typically, the input to a `Dense` layer is a 1D array, but in this example, the input is a 2D array with a shape of (430, 3). Since the model is composed entirely of `Dense` layers, executing each test case results in 430 runs, and the output is a 2D output array of shapes (430, 3). In other words, each test case produces $430 \times 3 = 1290$ output values. We examined the activation states of neurons for each test case during its execution. From the perspective of individual test cases, a portion of the neurons in the last layer are activated every time a test case runs. From the perspective of individual runs, out of 100 test cases with 43000 runs, there were 35699 instances where at least one of the three neurons in the last layer was activated.

> Answer to Review_B_Q1: In the last layer of the model, there are some neurons count as activated.

*C. Answer to **Review_B_Q2**: Can you re-instate the SBFL intuition, but for DNN models? How can we be sure that layer level model faults listed in Table II will always result in activation of these neurons in incorrect inferences?*

We argue that there may not be a direct correlation between the behavior of a DL model and the program used for its implementation. This means that the behavior of the DL model cannot be explicitly encoded in the program's control flow structures. Moreover, test cases for deep learning software are executed on the model, where the basic unit, the neurons, cannot be directly represented in the code as statements are. This creates a gap between the neuron-level of the model and the layer-level in the program. Thus, SBFL cannot be directly applied to DL programs. Spectrum needs to be constructed at the neuron-level rather than at the statement-level.

We observe a certain relationship between neuron activation states and model errors. For example, incorrect activation functions can impact the output range of neurons and the output distribution of the layer, thereby affecting weight updates during model training, which in turn affects the activation states. Improper weight initialization might cause certain neurons to remain permanently activated or suppressed during training, preventing them from effectively learning features. These errors will manifest as abnormal activation patterns and incorrect inferences.

The following example demonstrates a classification model:

```
model = Sequential()
model.add(Dense(2, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(2, activation='softmax')) # relu
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='
    binary_crossentropy', metrics=['accuracy'])
```

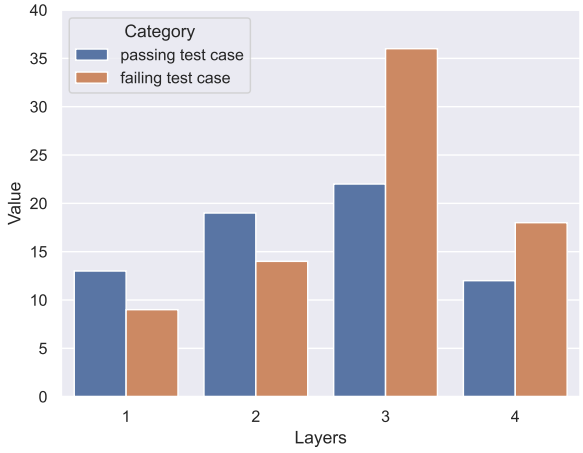

Fig. 2. The total number of neurons activated by all passing and failing test cases in each layer.

In this example, due to the improper activation function setting in the third layer, the model has low accuracy. The testing set consists of 30 test cases, with 12 passing and 18 failing test cases. We examined the activation states of neurons for each test case across all layers and calculated the total number of active neurons for all passing and failing test cases separately in each layer. The results are shown in Figure 2. It can be seen that for failing test cases, the number of activated neurons in the third layer is higher compared to the other layers. The output of GUIDER(Ochiai) is `[0.295, 0.162, 0.802, 0.798]`, indicating that the third layer exhibits the highest suspiciousness value.

> Answer to Review_B_Q2: There is a certain relationship between neuron activation states and model errors.

*D. Answer to **Review_B_Q3**: Please explain the overall manual inspection process for the studied bugs, as well as why rows 6 and 14 are considered model bugs.*

The studied bugs come from two sources: one part is from previous research, and for the other part, we initially collected candidate bugs from Stack Overflow using SQL query, resulting in the `QueryResults.csv` file. We then conducted a further manual review to identify the actual model bugs, resulting in the `dataset.csv` file. As for rows 6 and 14 in file `QueryResults.csv`, we do not consider them as model errors. We provide partial screenshots of files `QueryResults.csv` and `dataset.csv` to illustrate this point.

Figure 3 shows a screenshot of `QueryResults.csv`. The post ID corresponding to row 6 is 77536125, and the post ID corresponding to row 14 is 77250357. These are marked with red lines in the screenshot. Figure 4 shows a partial screenshot of `dataset.csv`. This file is our final dataset, which includes information such as post ID, whether it is a

	title	
2	Float16 mixed precision being slower than regular float32, keras, tensorflow 2.0	https://stackoverflow.com/questions/77717432
3	Why is pytorch loss unstable when reaching minimum, while keras loss keeps stable?	https://stackoverflow.com/questions/77672172
4	Extracting an item from a TensorFlow InputLayer	https://stackoverflow.com/questions/77616720
5	Manually perform predictions using the model weights and biases (Keras)	https://stackoverflow.com/questions/77576284
6	Plot the SHAP values for LSTM model in Python	https://stackoverflow.com/questions/77536125
7	Custom metric function for TensorFlow doesn't work correctly, it's always outputs zero	https://stackoverflow.com/questions/77524363
8	Advanced Machine Learning in Python: Handling Class Imbalance in Multi-class Classification with Custom Loss Function	https://stackoverflow.com/questions/77462336
9	Low recall and f1-score for LSTM Text classification	https://stackoverflow.com/questions/77455103
10	Keras mean squared error not working over a 3-dimensional response	https://stackoverflow.com/questions/77411040
11	Deep Learning with Python (Keras): Reuters Multiclass Classification	https://stackoverflow.com/questions/77338685
12	keras.layers.Dense not assigning correct names for layers. Instead getting "module_wrapper_x" names for each layer	https://stackoverflow.com/questions/77320188
13	Save keras model and StandartScaller to TensorflowJS	https://stackoverflow.com/questions/77300592
14	Discrepancies between the plot and the historical data object values of loss and valid_loss	https://stackoverflow.com/questions/77250357

Fig. 3. A partial screenshot of file `QueryResults.csv` (lines 1 to 14)

1	id	is_classification	groundtruth	error_type
2	74669249	0	11	1
3	74200607	0	0,1,2	3
4	73567385	1	9	4
5	73308371	1	3	1
6	73275569	1	9	3
7	72965428	1	5	1
8	72802464	1	3	3
9	72744278	1	5	3
10	72676542	1	13	1

Fig. 4. A partial screenshot of file `dataset.csv` (lines 1 to 10)

classification model, the number of error layers, and error type. The post IDs are sorted in descending order. As can be seen, the post ID in line 2 is 74669249, which is smaller than both 77536125 and 77250357, indicating that our dataset does not contain these two bugs.

> Answer to Review_B_Q3: Rows 6 and 14 do not considered as model bugs in our dataset.

*E. Answer to **Review_B_Q4**: Is three retrainings and three samplings really enough? Can you provide statistical evidence?*

We conducted 9 (3 retraining $\times$ 3 samplings) repeated experiments for two reasons. First, we observed that the results of each repeated experiment showed relatively small differences. Second, our dataset includes some large-scale models, and conducting too many repeated experiments would result in significant overhead. Considering these two aspects, we ultimately decided to set the number of repeated experiments to 9. We also calculated the variance of the results from 9 repeated experiments.

Table I shows the mean and variance of the number of bugs detected by our method in 9 repeated experiments. In the last column of rows 7 to 11, we added the minimum and maximum of the total number in parentheses. We can see that both the total number and the number for each category have relatively small variances across the 9 repeated experiments.

TABLE I
MEAN AND VARIANCE OF THE NUMBER OF BUGS DETECTED BY OUR METHOD IN 9 REPEATED EXPERIMENTS.

Metrics	Tool	Cat.1	Cat.2	Cat.3	Cat.4	Total (detected)
Mean	GUIDER(Tarantula)	72.67	2.56	4.89	0.00	80.11
	GUIDER(Ochiai)	71.00	2.89	6.33	0.00	80.22
	GUIDER(Dstar)	67.56	2.89	5.89	0.00	76.33
	GUIDER(Op2)	70.00	1.89	6.33	0.00	78.22
	GUIDER(Barinel)	71.67	3.22	5.00	0.00	79.89
Variance	GUIDER(Tarantula)	1.75	0.53	0.61	0.00	5.86 (77-83)
	GUIDER(Ochiai)	0.75	0.36	1.00	0.00	1.19 (79-82)
	GUIDER(Dstar)	1.53	0.36	1.11	0.00	2.25 (74-78)
	GUIDER(Op2)	3.00	0.36	0.25	0.00	0.94 (77-80)
	GUIDER(Barinel)	0.25	0.19	0.75	0.00	0.61 (79-81)

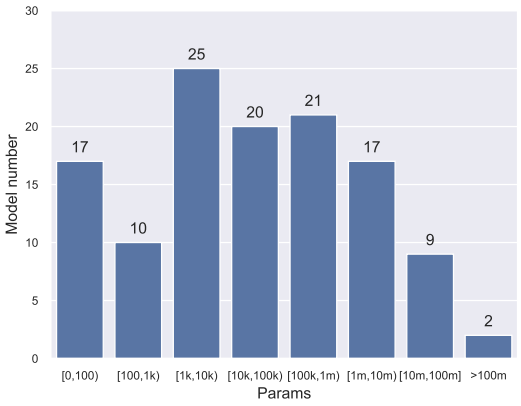

Fig. 5. The number of models with different parameter ranges.

This indicates that the results of each repeated experiment are quite consistent, suggesting that 9 repetitions are enough.

> Answer to Review_B_Q4: The small variance in the results of the 9 repeated experiments indicates that the number of repetitions is enough.

*F. Answer to **Review_C_Q1**: What about the scale and complexity of the studied error models?*

To introduce the details of the studied models, we collected statistics on the number of layers, parameters, and architecture types for each model. The detailed information can be found in the `model_info.csv` file in our GitHub repository. We also calculated some statistical information in this RQ.

Figure 5 shows the number of models with different parameter ranges. It can be seen that the models we studied cover a wide range of sizes. The smallest model has fewer than 100 parameters, while the largest model has over 100 million parameters. Figure 6 shows the number of models with different layers. The studied models also exhibit a wide distribution in terms of the number of layers. Figure 7 presents the number and proportion of models with different architecture types. Among our studied models, there are 64 fully-connected

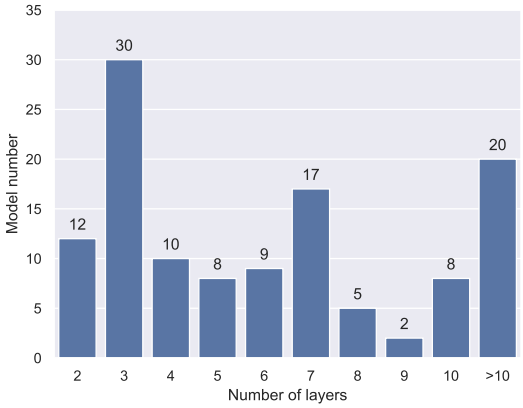

Fig. 6. The number of models with different layers.

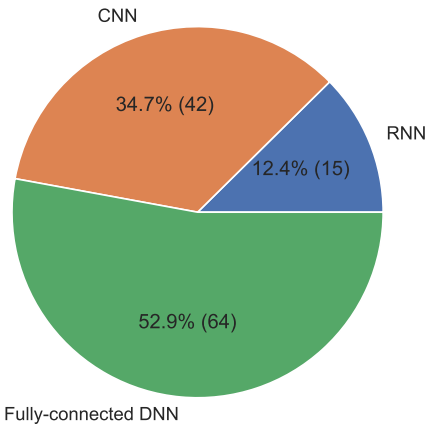

Fig. 7. The number and proportion of models with different architecture types.

DNNs, 15 RNNs, and 42 CNNs, accounting for 52.9%, 12.4%, and 34.7%, respectively.

> Answer to Review_C_Q1: Our studied models encompassed various scales and complexities.