# Silent Bugs in Keras and TensorFlow

In this survey, we explore opinions of researchers/practitioners about silent bugs in Deep Learning libraries.

* Required

1. Your email (optional)

2. What is your current job title? (Developer, Researcher, ... students may indicate degree: PhD, Master)

3. Year of experiment using Keras/Tensorflow *

*Mark only one oval.*

- ◯ Less than 1 year
- ◯ 1-3 years
- ◯ 3-5 years
- ◯ More than 5 years

## Notion of Silent Bugs

4. Is the term of "Silent Bug" familiar to you? *

*Mark only one oval.*

- ◯ Yes
- ◯ No

**In our study, we define Silent Bugs as "any bug that did not stop the program, for example not causing the program to crash, hang (with or without an error message) or fail. Bugs that lead to memory leakage are not considered as silent." Note: these bugs are residing of Tensorflow/Keras library, NOT inside user's program.**

5. Is this definition meaningful to you? *

*Mark only one oval.*

- ◯ Yes
- ◯ No

6. Any comments?

## Categorization of Scenario

Based on extracted existing issues, we came up with a classification of silent bugs based on how they affected user's program. In this section, you will be asked to assess those different category regarding three criteria. Every time, you will be given the name of the category and a short description.

The criteria are:
- Diagnosing: How hard it is to detect such bugs?
- Severity: How impactful the bug is regarding user's program?
- Fixing: How hard was it to fix the bug?

You will then be ask whether or not you ever faced something matching this category. Remember, bugs must have not produced any error, hang or failure and must have originated from the TensorFlow/Keras library itself!

Please, even if you never came across such bug, answer based on your own experience.

7. 1. Wrong Shape: This category includes any bug leading to a wrong shape of a tensor in the model without raising an error. In the following sample: Keras returns bad shape for the user's custom layer. The output given by Keras differs from the compute_output_shapefunction of the custom Layer. *

```
import tensorflow as tf
import numpy as np

class Example(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        kwargs["dynamic"] = True
        super(Example, self).__init__(**kwargs)

    def call(self, inputs):
        return inputs

    def compute_output_shape(self, input_shape):
        return [(None, 2)]

inp = tf.keras.layers.Input(batch_shape=(None, 1))
comp = Example()(inp)

model = tf.keras.models.Model(inputs=[inp], outputs=[comp])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
example (Example)            [(None, (2,))]            0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
```

*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

8. Have you ever been confronted to such type of bugs? *

*Mark only one oval.*

- ◯ Yes
- ◯ No

9. Any comments?

10. 2. Wrong Displayed Message: Any bug that shows an information in UI (including console messages) which will deceive the user or affect the user understanding of the ML model belongs to this category. In the following sample: Keras displays a progress bar that is too long and which eventually masks part of the UI. The progress bar given by Keras is too long between train and test phase. The error was linked to a wrong variable being passed to the iteration routine used in the training/evaluation loop of the API. *

```
import tensorflow as tf

mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) =
       mnist.load_data()
training_images = training_images / 255.0
test_images = test_images / 255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
       metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5)

test_loss = model.evaluate(test_images, test_labels)
```

```
Train on 60000 samples
[...]
Epoch 5/5
60000/60000 [==============================] - 2s 38us/sample - loss: 0.2980 -
       accuracy: 0.8903
10000/1 [==============================================
...
... Literally hundreds of thousands of `=` ...
...
==========================================] - 0s 26us/sample - loss: 0.2803 -
       accuracy: 0.8673
```

*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

11. Have you ever been confronted to such type of bugs? *

*Mark only one oval.*

- ◯ Yes
- ◯ No

12. Any comments?

13. 3. Wrong Parameter Settings: Any bugs affecting the expected parameters setting of a function or model's components fall in this category. In the example below: passing a variable to learning rate and dynamically changing it will not be registered on the learning rate. Even though the learning rate variable got set to 0, the variable of the trainable layer is still being modified. *

```
import tensorflow as tf
print(tf.version.GIT_VERSION, tf.version.VERSION)
import sys
print(sys.version_info)
tf_a = tf.Variable(1.0)
print('Variable tf_a initialized to {}.'.format(tf_a.numpy()))
tf_lr = tf.Variable(0.1, trainable=False)
tf_opt = tf.keras.optimizers.Adam(learning_rate=tf_lr)
@tf.function
def train_step():
    with tf.GradientTape() as tf_tape:
        tf_loss = tf_a**2
    tf_gradients = tf_tape.gradient(tf_loss, [tf_a])
    tf_opt.apply_gradients(zip(tf_gradients, [tf_a]))
print('After one step with learning rate {}... '.format(tf_lr.numpy()), end='')
train_step()
print('Variable tf_a is {}.'.format(tf_a.numpy()))
tf_lr.assign(0.0)
for _ in range(10):
    print('After another step, now with learning rate {}...
          '.format(tf_lr.numpy()), end='')
    train_step()
    print('Variable tf_a is {}.'.format(tf_a.numpy()))
```

```
v2.0.0-beta0-16-g1d91213fe7 2.0.0-beta1
sys.version_info(major=3, minor=5, micro=6, releaselevel='final', serial=0)
Variable tf_a initialized to 1.0.
After one step with learning rate 0.10000000149011612... Variable tf_a is
       0.8999971747398376.
After another step, now with learning rate 0.0... Variable tf_a is
       0.8004083633422852.
After another step, now with learning rate 0.0... Variable tf_a is
       0.7015821933746338.
[...]
After another step, now with learning rate 0.0... Variable tf_a is
       0.07624538242816925.
After another step, now with learning rate 0.0... Variable tf_a is
       0.005127914249897003.
```

*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

14. Have you ever been confronted to such type of bugs? *

3. Wrong Parameter Settings: Any bugs affecting the expected parameters setting of a function or model's components fall in this category. In the example below: passing a variable to learning rate and dynamically changing it will not be registered on the learning rate. Even though the learning rate variable got set to 0, the variable of the trainable layer is still being modified.

*Mark only one oval.*

- ◯ Yes
- ◯ No

15. Any comments?

16. 4. Wrong save/reload: In this category, we classify all bugs that change the model, its component or its functionalities either during saving or re-loading. For example, missing a layer after reload or inconsistent accuracy before and after saving. In the following example: When reloading the model, the accuracy of the model on the same data drastically fell. Using the weights without verification, the model would return mostly incorrect predictions. *

```
import tensorflow as tf

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=2, activation='softmax', name='output'))
model.compile(optimizer=tf.keras.optimizers.Adam(lr=10),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
dummy_data_x = [[0, 0], [1, 0], [0, 1], [1, 1]]
dummy_data_y = [0, 1, 0, 1]
print(model.evaluate(x=dummy_data_x, y=dummy_data_y))
model.fit(x=dummy_data_x, y=dummy_data_y, epochs=10)
print(model.evaluate(x=dummy_data_x, y=dummy_data_y))
model.save('test_model')
model = tf.keras.models.load_model('test_model')
print(model.evaluate(x=dummy_data_x, y=dummy_data_y))
```

```
Before training:
1/1 [==============] - 0s 0s/step - loss: 0.9013 - accuracy: 0.5000
[0.9013183116912842, 0.5]

After training:
1/1 [==============] - 0s 0s/step - loss: 0.0000e+00 - accuracy: 1.0000
[0.0, 1.0]

After loading:
1/1 [==============] - 0s 1000us/step - loss: 0.0000e+00 - accuracy: 0.5000
[0.0, 0.5]
```

*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

17. Have you ever been confronted to such type of bugs? *

4. Wrong save/reload: In this category, we classify all bugs that change the model, its component or its functionalities either during saving or re-loading. For example, missing a layer after reload or inconsistent accuracy before and after saving. In the following example: When reloading the model, the accuracy of the model on the same data drastically fell. Using the weights without verification, the model would return mostly incorrect predictions.

*Mark only one oval.*

- ◯ Yes
- ◯ No

18. Any comments?

19. 5. Performance Degradation: Any bug affecting the performance of ML experiments (training or inference) is categorized in this class, e.g., memory usage or running time (speed). This category does not include changes in prediction accuracy of the model. An example: consecutive calls to either fit() or evaluate() increases the used main memory (RAM) even when calling with the same data. The user complains that such calls take approximately ten times longer than with TF1.x. *

```
from memory_profiler import profile
from time import time
import numpy as np
import tensorflow as tf

model = tf.keras.Sequential([tf.keras.layers.Dense(100,
       activation=tf.nn.softmax)])
model.compile(loss='mse', optimizer='sgd')
@profile
def eval(x, y):
    model.evaluate(x, y)
x = np.random.normal(size=(1,100))
y = np.random.normal(size=(1,100))
for i in range(100000):
    print('iteration', i)
    tic = time()
    eval(x, y)
    print('timeit', time() - tic)
```

```
iteration 3312
1/1 [==============================] - 0s 4ms/sample - loss: 1.0205
Filename: reproduce_keras_oom.py

Line    Mem usage    Increment  Line Contents
================================================
     9   1508.3 MiB  1508.3 MiB  @profile
    10                          def eval(x, y):
    11   1508.7 MiB     0.4 MiB      model.evaluate(x, y)

timeit 0.09004998207092285
```

*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

20. Have you ever been confronted to such type of bugs? *

5. Performance Degradation: Any bug affecting the performance of ML experiments (training or inference) is categorized in this class, e.g., memory usage or running time (speed). This category does not include changes in prediction accuracy of the model. An example: consecutive calls to either fit() or evaluate() increases the used main memory (RAM) even when calling with the same data. The user complains that such calls take approximately ten times longer than with TF1.x.

*Mark only one oval.*

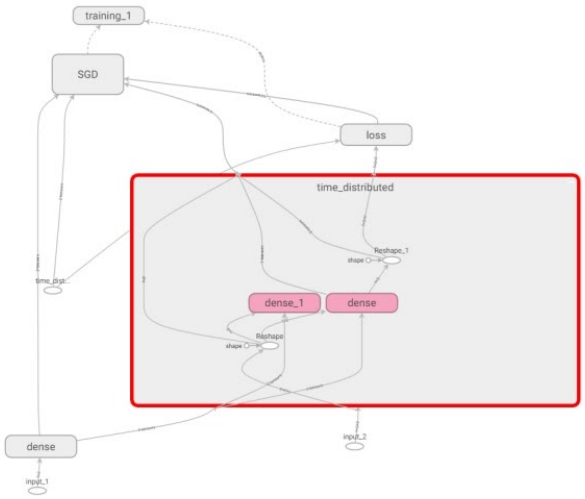
- ◯ Yes
- ◯ No

21. Any comments?

22. 6. Wrong Structure: The category covers all bugs that modify the expected structure of a model, in particular how it is handled by the framework. In the following example: the user attempts to wrap a model in a TimeDistributed layer but this leads to creation of duplicate nodes in the graph. Following the documentation of the framework, the user ends up with an additional Dense Layer (bottom left in the figure). This additional layer takes redundant memory and is created because the user builds the inner model then rebuilds it again during building the TimeDistributed model. *

```
inner_input = keras.layers.Input((2,))
dense = keras.layers.Dense(2, activation='relu')(inner_input)
inner_model = keras.Model(inputs=inner_input, outputs=dense)
full_input = keras.layers.Input((2,2))
td_2 = keras.layers.TimeDistributed(inner_model)(full_input)
model = keras.models.Model(full_input, td_2)
model.compile('SGD', 'mse')
```

model:




*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

23. Have you ever been confronted to such type of bugs? *

6. Wrong Structure: The category covers all bugs that modify the expected structure of a model, in particular how it is handled by the framework. In the following example: the user attempts to wrap a model in a TimeDistributed layer but this leads to creation of duplicate nodes in the graph. Following the documentation of the framework, the user ends up with an additional Dense Layer (bottom left in the figure). This additional layer takes redundant memory and is created because the user builds the inner model then rebuilds it again during building the TimeDistributed model.

*Mark only one oval.*

◯ Yes
◯ No

24. Any comments?

25. 7. Wrong Calculation: All bugs modifying the normal way of computation that are not classified in other categories will be classified in this type, like wrong calculation of gradient. In the displayed bug, the evaluate() function computes the mean loss over all batches in an epoch incorrectly when the dataset size is not evenly divisible by the batch size. This happens for both training and validation loss. Actually, the bug affects the reported epoch loss, but not the training loss used for computing gradient updates. In the provided gist, there are 3 samples in the dataset, and the batch size is 2. So, there are 2 batches of size 2 and 1. If the first batch has mean loss of 10 and the second batch has mean loss of 9, then the mean loss over the entire dataset is incorrectly computed as (10 + 9) / 2 = 9.5. *

```
import tensorflow as tf

X = tf.constant([[1], [2], [3]], dtype=tf.float32)
y = tf.constant([[5], [4], [6]], dtype=tf.float32)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_dim=1, kernel_initializer='ones',
           bias_initializer='zeros')])
model.compile(optimizer='sgd', loss='mean_squared_error')
def mse(y, y_pred):
    assert len(y) == len(y_pred)
    return sum((y - y_pred)**2)/len(y)
print('model.evaluate():')
print('- batch_size=1:', model.evaluate(X, y, batch_size=1, verbose=0))
print('- batch_size=2:', model.evaluate(X, y, batch_size=2, verbose=0))
print('- batch_size=3:', model.evaluate(X, y, batch_size=3, verbose=0))
print()
print((mse(X[:-1], y[:-1]) + mse(X[-1], y[-1]))/2)
```

```
model.evaluate():
- batch_size=1: 9.666666984558105
- batch_size=2: 9.5
- batch_size=3: 9.666666984558105

tf.Tensor([9.5], shape=(1,), dtype=float32)
```

*Mark only one oval per row.*

| | 1 (easy) | 2 | 3 | 4 | 5 (difficult) |
|---|---|---|---|---|---|
| Diagnosing | ◯ | ◯ | ◯ | ◯ | ◯ |
| Severity | ◯ | ◯ | ◯ | ◯ | ◯ |
| Fixing | ◯ | ◯ | ◯ | ◯ | ◯ |

26. Have you ever been confronted to such type of bugs? *

7. Wrong Calculation: All bugs modifying the normal way of computation that are not classified in other categories will be classified in this type, like wrong calculation of gradient. In the displayed bug, the evaluate() function computes the mean loss over all batches in an epoch incorrectly when the dataset size is not evenly divisible by the batch size. This happens for both training and validation loss. Actually, the bug affects the reported epoch loss, but not the training loss used for computing gradient updates. In the provided gist, there are 3 samples in the dataset, and the batch size is 2. So, there are 2 batches of size 2 and 1. If the first batch has mean loss of 10 and the second batch has mean loss of 9, then the mean loss over the entire dataset is incorrectly computed as (10 + 9) / 2 = 9.5.

*Mark only one oval.*

◯ Yes
◯ No

27. Any comments?

28. Have you observed any silent bug in DL libraries that have not been considered in this survey? If yes, please describe them:

Categorization of impact

Following categories design, we came up with a scale to rate impact level of such silent bugs based on the effect they had on the user's program. We divided it in 4 different levels, with gradual increasing threats.

You will be ask to assess your agreement with the threat level (1, 2, 3 or 4) based on the description we provide for each level.

Finally, we will give you a representation of the scale, and we ask you to assess if the global scale representation is understandable and correct according to you.

29. Level 1: The issue impacts the user interface (UI) element, either in a purely cosmetic way or with little to no impact on the information the user receives. Like long progress bar: *

```
import tensorflow as tf

mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) =
      mnist.load_data()
training_images = training_images / 255.0
test_images = test_images / 255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
      metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5)

test_loss = model.evaluate(test_images, test_labels)
```

```
Train on 60000 samples
[...]
Epoch 5/5
60000/60000 [==============================] - 2s 38us/sample - loss: 0.2980 -
      accuracy: 0.8903
10000/1 [==============================================
...
... Literally hundreds of thousands of `=` ...
...
==========================================] - 0s 26us/sample - loss: 0.2803 -
      accuracy: 0.8673
```

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly agree |

30. Any comments?

31. Level 2: The issue impacts information the user is receiving from the model, e.g., shape of a model, loss value... but does not actually modify the way the model works or its results. For example, wrong shape: *

```
import tensorflow as tf
import numpy as np

class Example(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        kwargs["dynamic"] = True
        super(Example, self).__init__(**kwargs)

    def call(self, inputs):
        return inputs

    def compute_output_shape(self, input_shape):
        return [(None, 2)]

inp = tf.keras.layers.Input(batch_shape=(None, 1))
comp = Example()(inp)

model = tf.keras.models.Model(inputs=[inp], outputs=[comp])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
example (Example)            [(None, (2,))]            0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
```

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly agree |

32. Any comments?

33. Level 3: The issue affects the way the model works and its process, e.g., hyper-parameters, layer composition, or model processing mechanism which can lead to some time/speed degradation but only minor to no changes on the model outputs. For example, wrongly set learning rate: *

```
import tensorflow as tf
print(tf.version.GIT_VERSION, tf.version.VERSION)
import sys
print(sys.version_info)
tf_a = tf.Variable(1.0)
print('Variable tf_a initialized to {}.'.format(tf_a.numpy()))
tf_lr = tf.Variable(0.1, trainable=False)
tf_opt = tf.keras.optimizers.Adam(learning_rate=tf_lr)
@tf.function
def train_step():
    with tf.GradientTape() as tf_tape:
        tf_loss = tf_a**2
    tf_gradients = tf_tape.gradient(tf_loss, [tf_a])
    tf_opt.apply_gradients(zip(tf_gradients, [tf_a]))
print('After one step with learning rate {}... '.format(tf_lr.numpy()), end='')
train_step()
print('Variable tf_a is {}.'.format(tf_a.numpy()))
tf_lr.assign(0.0)
for _ in range(10):
    print('After another step, now with learning rate {}...
          '.format(tf_lr.numpy()), end='')
    train_step()
    print('Variable tf_a is {}.'.format(tf_a.numpy()))
```

```
v2.0.0-beta0-16-g1d91213fe7 2.0.0-beta1
sys.version_info(major=3, minor=5, micro=6, releaselevel='final', serial=0)
Variable tf_a initialized to 1.0.
After one step with learning rate 0.10000000149011612... Variable tf_a is
       0.8999971747398376.
After another step, now with learning rate 0.0... Variable tf_a is
       0.8004083633422852.
After another step, now with learning rate 0.0... Variable tf_a is
       0.7015821933746338.
[...]
After another step, now with learning rate 0.0... Variable tf_a is
       0.07624538242816925.
After another step, now with learning rate 0.0... Variable tf_a is
       0.005127914249897003.
```

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly agree |

34. Any comments?

35. Level 4: The issue directly affects the way the model behaves which leads to completely different results compared to known results or the fixed version. Like a wrong model reloading: *

```
import tensorflow as tf

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=2, activation='softmax', name='output'))
model.compile(optimizer=tf.keras.optimizers.Adam(lr=10),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
dummy_data_x = [[0, 0], [1, 0], [0, 1], [1, 1]]
dummy_data_y = [0, 1, 0, 1]
print(model.evaluate(x=dummy_data_x, y=dummy_data_y))
model.fit(x=dummy_data_x, y=dummy_data_y, epochs=10)
print(model.evaluate(x=dummy_data_x, y=dummy_data_y))
model.save('test_model')
model = tf.keras.models.load_model('test_model')
print(model.evaluate(x=dummy_data_x, y=dummy_data_y))
```

```
Before training:
1/1 [==============] - 0s 0s/step - loss: 0.9013 - accuracy: 0.5000
[0.9013183116912842, 0.5]

After training:
1/1 [==============] - 0s 0s/step - loss: 0.0000e+00 - accuracy: 1.0000
[0.0, 1.0]

After loading:
1/1 [==============] - 0s 1000us/step - loss: 0.0000e+00 - accuracy: 0.5000
[0.0, 0.5]
```

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly agree |

36. Any comments?

Impact level scale: “X” means that to be considered of this impact level, a bug needs to meet those criteria. “O” denotes potential criteria that an issue can also have. Note that only the “X” criteria are needed to be considered of a given level while “O” criteria are optional.

| Impact | Level | | | |
|---|---|---|---|---|
| | Low 1 | 2 | 3 | High 4 |
| Does the bug impact a UI element? | X | O | O | O |
| Does the bug impact information about the model? | | X | O | O |
| Does the bug alter part of the model operation? | | | X | O |
| Does the bug change the result of the model? | | | | X |

37. How would you judge this scale? (presentation, comprehension, relevance wise) *

*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly agree |

38. Any comments?

Thank you very much for your answers!

39. Based on the previous questions and your experience, how problematic would you consider silent bugs to be compared to traditional ones (i.e. the ones returning error, leading to program hanging on, ...) *

*Mark only one oval.*

- ◯ Less problematic
- ◯ The same as others
- ◯ More problematic

40. Thank you for helping us with your feedback! Feel free to give any comments regarding the previous sections and the study in general (ideas? critics?...). Anything is welcomed!

This content is neither created nor endorsed by Google.

Google Forms