Differential Testing of Cross Deep Learning Framework APIs: Revealing Inconsistencies and Vulnerabilities

Zizhuang Deng\textsuperscript{1,2}, Guozhu Meng\textsuperscript{1,2,*}, Kai Chen\textsuperscript{1,2,*}, Tong Liu\textsuperscript{1,2}, Lu Xiang\textsuperscript{1,2}, Chunyang Chen\textsuperscript{3}

\textsuperscript{1} SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China
\textsuperscript{2} School of Cyber Security, University of Chinese Academy of Sciences, China
\textsuperscript{3} Monash University, Australia

The 32nd USENIX Security Symposium (USENIX Security 2023)
Background

• Deep learning systems are rapidly evolving
Cross Deep learning Framework APIs

Using differential testing for cross deep learning framework testing.
Differential Testing is **challenging**

- Extract the constraints of API parameters and their implicit dependencies
Differential Testing is **challenging**

- Extract constraints
- Generate representative test cases
It’s hard to **evaluate** these bugs

- APIs in Real world models
- Craft model input to trigger the buggy API
Counterpart APIs

• For an API $f$, $\text{counterpart}(f) = \{f_1, \ldots, f_n\}, (n \geq 1)$

  - Semantic equivalence

    $\|f(x) - (f_1 \circ \ldots \circ f_n)(x)\|_p \leq \varepsilon$

  - Sequentiality

    • AdjustContrastv2($x_0, x_1$) = Add(Mul($x_1$, Sub($x_0$, ReduceMean($x_0$))), ReduceMean($x_0$))

How to extract counterpart APIs across DL frameworks?
Counterparts Extraction

- Model converter

Diagram:
- TensorFlow
  - tf2onnx
  - onnx2tf
- ONNX
  - onnx2torch
  - torch.onnx.export
- PyTorch
Counterparts Extraction

• Static analysis on converter code

```python
registry: Dict[str, handler] = {
"onnx::AveragePool": PoolMapper,
"onnx::MaxPool": PoolMapper, ...
}
class PoolMapper(ONNXToMindSporeMapper):
def _operation_name_in_ms(*args, **kwargs):
    if kwargs['op_name'] == 'onnx::AveragePool':
        op_name = "nn.AvgPool{}d"
    else:
        op_name = "nn.MaxPool{}d"
    dim = len(kwargs['params']['strides'])
    if dim == 3:
        return "P.MaxPool3D"
    return op_name.format(dim)
```
Counterparts Extraction

- Parameters Alignment
Constraint Extraction

- Constraints of API parameters
  - On five attributes: ① type ② shape ③ data type ④ rank ⑤ data value

- API profiles
Constraint Extraction

• API Implementation

Example:

```c
OP_REQUIRES(
  ctx, a_shape->IsSameSize(*b_shape), ...);
```
Test Case Generation

• Joint Constraints Analysis
  • For the rank attribute of parameter
    \( C_p^A / C_p^B \) represents constraints in framework A/B

Using intersection set to test deeper code.

Using difference set to test corner-case code.
Testing Optimization

• Error-guided Test Case Fixing

This optimization is to test deeper code.
Testing Optimization

• Range Extension

Value: [min, max]

Shape: (2, 3) → (2, 3, 1)

Special values, differential parameters

\[
\text{mid} = \frac{\text{max} - \text{min}}{2}
\]

This optimization is to test corner-case code.
Evaluation on bug finding

- Statistics of crash bugs(177) & non-crash bugs(80)
  - 230 ones are newly found

<table>
<thead>
<tr>
<th>Version</th>
<th>#Bug</th>
<th>Segv</th>
<th>FPE</th>
<th>Abort</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF</td>
<td>2.11</td>
<td>26</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>TFL</td>
<td>2.11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ORT</td>
<td>1.12.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MS</td>
<td>1.9.0 &amp; nightly</td>
<td>100</td>
<td>90</td>
<td>8</td>
</tr>
<tr>
<td>Paddle</td>
<td>develop</td>
<td>23</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>PyTorch</td>
<td>1.10.0 &amp; 1.12.1</td>
<td>28</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>177</td>
<td>145</td>
<td>14</td>
<td>18</td>
</tr>
</tbody>
</table>

(a) Crash bugs

(b) Inconsistent (non-crash) bugs

We found a total of 257 bugs on 1,658 APIs of 6 DL frameworks.
Evaluation on bug finding

• 8 CVEs
• $1,100+ bounty!

<table>
<thead>
<tr>
<th>ID</th>
<th>CVSS</th>
<th>Framework</th>
<th>Type</th>
<th>Symptom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2022-35935</td>
<td>7.5</td>
<td>TensorFlow</td>
<td>missing validation</td>
<td><code>CHECK</code> failure</td>
<td>given a non scalar <code>num_results</code> value</td>
</tr>
<tr>
<td>CVE-2022-41883</td>
<td>7.5</td>
<td>TensorFlow</td>
<td>missing validation</td>
<td>OOB segfault</td>
<td><code>indices</code> list shorter than the <code>data</code> list</td>
</tr>
<tr>
<td>CVE-2022-41899</td>
<td>7.5</td>
<td>TensorFlow</td>
<td>missing validation</td>
<td>segfault</td>
<td>given wrong shape tensors</td>
</tr>
<tr>
<td>CVE-2022-41891</td>
<td>7.5</td>
<td>TensorFlow</td>
<td>missing validation</td>
<td>segfault</td>
<td>element_shape=[]</td>
</tr>
<tr>
<td>CVE-2022-41897</td>
<td>7.5</td>
<td>TensorFlow</td>
<td>missing validation</td>
<td>Heap OOB</td>
<td>outsize inputs</td>
</tr>
<tr>
<td>CVE-2022-45907</td>
<td>9.8</td>
<td>PyTorch</td>
<td>code injection</td>
<td>arbitrary code execution</td>
<td>using dangerous <code>eval</code></td>
</tr>
<tr>
<td>CVE-2022-45908</td>
<td>9.8</td>
<td>Paddle</td>
<td>code injection</td>
<td>arbitrary code execution</td>
<td>using dangerous <code>eval</code></td>
</tr>
<tr>
<td>CVE-2022-46742</td>
<td>9.8</td>
<td>Paddle</td>
<td>code injection</td>
<td>arbitrary code execution</td>
<td>using dangerous <code>eval</code></td>
</tr>
</tbody>
</table>
Evaluation on code coverage

- Comparative experiments
- Compare with previous work
- Ablation study

It is observed that our tool (TensorScope) detects the most number of bugs and the highest code coverage. These results indicate that joint constraints can effectively guide the testing process.
Case study of converter bugs

```python
AdjustContrastv2(
    Images=tf.random.uniform([64,64,3,2],dtype=tf.dtypes.float32,maxval=255),
    contrast_factor=tf.random.uniform([],dtype=tf.dtypes.float32,maxval=1),
)

Here is the inconsistent results:

tf_res : [[[154.93831, 131.07579], [141.15346, 162.48589], [123.68347, 143.64304]], ... ]

onnx_res : [[[153.70927, 126.60315], [139.92442, 158.01324], [122.45444, 139.1704]], ... ]]
Hazard analysis

- **MobileNet model**
  - Top-1 accuracy 72.3% (TensorFlow) -> 68.8% (ONNX)
- **Classify “Snail” to “bubble”**
Thanks!

Q&A

Contact: dengzizhuang@iie.ac.cn