BinFl: An Efficient Fault Injector for Safety-Critical Machine Learning Systems

Zitao Chen, Guanpeng Li, Karthik Pattabiraman, Nathan DeBardeleben
Motivation

• Machine learning taking computing by storm
  • HPC-ML: precision medicine; earthquake simulation;
  • Growing safety-critical applications.
• Reliability of ML becomes important.

Soft Error

- Transient hardware fault
  - Growing in frequency:
    - Occur every 53 mins in 1m nodes [1]
  - Manifested as a single bit-flip
- Silent data corruptions (SDCs)
  - Erroneous ML output.
- Safety standard for road vehicles:
  - ISO26262: 10 FIT

Existing solutions

- **Application-agnostic:**
  - Triple modular duplication (TMR) for execution units.
  - Expensive: Hardware cost, performance (e.g., delay-based accidents).

- **Application-specific:**
  - *Random fault injection* to guide protection (e.g., instruction duplication).
  - Coarse-grained
Is Random Flip Good Enough?

Randomly simulate bit-flip, and then obtain statistical error resilience

1. Where are the *critical faults* in the entire system?
2. Are the critical faults uniformly distributed (or not)?
Our goal

An efficient approach to obtain fine-grained understanding of the error resilience of ML systems
Contributions:

• Identify the property of ML computations which constrain the fault propagation behaviors.

• Characterize the pattern of critical faults.

• Propose a Binary-FI approach to identify critical bits.
ML framework - TensorFlow

- TensorFlow: **framework for executing dataflow graphs**
- ML algorithms expressed as dataflow graphs

- Others: ![PyTorch](https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session) ![Caffe](https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session)

Image source: https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session
Fault model

- Focus on inference phase
- Faults at processor’s datapath (e.g. ALUs)
- Interface-level fault injection (i.e. TensorFlow Operators) [2]


Image source: https://medium.com/@d3lm/understand-tensorflow-by-mimicking-its-api-from-scratch-faa55787170d
How to cause an SDC in ML

• In ML, fault usually results in numerical change in the data.
• Output by ML is usually determined by numerical magnitude.
  
  *To cause SDC*: large deviation at the output

Large positive deviation

Large negative deviation
Error propagation

- Error propagation (EP): from the fault occurrence to the output.

- **Each EP function**: large response to Large input, i.e., monotone

- Convolution function: $X \ast W = \sum x_i w_i$
  - Larger Input deviation: $A > B$
  - Larger Output deviation: $|Aw_i| \geq |Bw_i|$

![Diagram of VGG16 architecture with annotations](image-url)
Individual EP function

- Common computations in DNNs, satisfy monotone property
- Why: ML tends generate large responses to "target" class/feature

<table>
<thead>
<tr>
<th>Basic</th>
<th>Conv; MatMul; Add (BiasAdd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation</td>
<td>ReLu; ELu;</td>
</tr>
<tr>
<td>Pooling</td>
<td>Max-pool; Average-pool</td>
</tr>
<tr>
<td>Normalization</td>
<td>Batch normalization (BN);</td>
</tr>
<tr>
<td></td>
<td>Local Response Normalization (LRN)</td>
</tr>
<tr>
<td>Data transformation</td>
<td>Reshape; Concatenate; Dropout</td>
</tr>
<tr>
<td>Others</td>
<td>SoftMax; Residual function</td>
</tr>
</tbody>
</table>
Error propagation example

• One fault propagates into multiple faults.

All EP functions

• Composite EP function:

  $EP(x) = 100 \times \max(x - 1, 0) - \max(x, 0)$

• Break of monotonicity: $EP(x)$

• We call it *approximate monotone*.
All EP functions (cont.)

- **Approximate** the EP behavior as an *approximate monotonic* function.

*Input (with faults) and output deviation are constrained by the approximate monotonicity*
Characteristic of critical faults

If a fault at high-order bits does not lead to SDC (by FI), faults at lower-order bits would not lead to SDC (without FI), i.e., SDC boundary.
Binary fault injection (BinFI)

- Consider the effects of different faults as a sorted array.

Search within a sorted array

Result in SDC
Move to lower-order bit

Not result in SDC
Move to higher-order bit
Experimental setup

- **SDC**: Image misclassification; degree of deviation.
- **Fault injection tool**: TensorFI
- **3 FI approaches**: BinFI vs Random FI vs Exhaustive FI

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dataset Description</th>
<th>ML model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNist</td>
<td>Hand-written digits</td>
<td>Neural Network LeNet</td>
</tr>
<tr>
<td>Survive</td>
<td>Prediction of patient survival</td>
<td>kNN</td>
</tr>
<tr>
<td>Cifar-10</td>
<td>General images</td>
<td>AlexNet</td>
</tr>
<tr>
<td>ImageNet</td>
<td>General images</td>
<td>VGG16</td>
</tr>
<tr>
<td>Traffic sign</td>
<td>Real-world traffic signs</td>
<td>VGG11</td>
</tr>
<tr>
<td>Driving</td>
<td>Real-world driving frames</td>
<td>Nvidia Dave Comma.ai</td>
</tr>
</tbody>
</table>

FI tool: [https://github.com/DependableSystemsLab/TensorFI](https://github.com/DependableSystemsLab/TensorFI)
Effects of SDCs

Single bit-flip can cause undesirable output in ML
Identification of critical bits

• **Recall**: 99.56% (average)
• **Precision**: 99.63% (average)

1. BinFI can identify most of the critical bits.
2. Random FI is not desirable for identifying critical bits.

![Graph showing comparison of recall and precision for different methods]

- **5x more overhead than BinFI**
- **Same overhead as BinFI**
Overhead

Overhead of BinFl is \( \sim 20\% \) of that by exhaustive Fl

- Data-width: 16, 32, 64 bits.
- 99.5+\% recall and precision

1. Overhead of BinFl grows *linearly* with the data width.
2. BinFl is *agnostic* to data width in identifying critical bits.
Summary

• Common ML functions exhibit monotonicity, which constrains the fault propagation behaviors.

• Critical faults in ML programs tend to be clustered: If a fault at high-order bit does not lead to SDC (by FI), faults at lower-order bits would not lead to SDC (without FI)

Zitao Chen, Graduate student at University of British Columbia
zitaoc@ece.ubc.ca

Artifact: https://github.com/DependableSystemsLab/TensorFI-BinaryFI