# Neural network based silent error detector

Chen Wang

Silent data corruption (SDC)

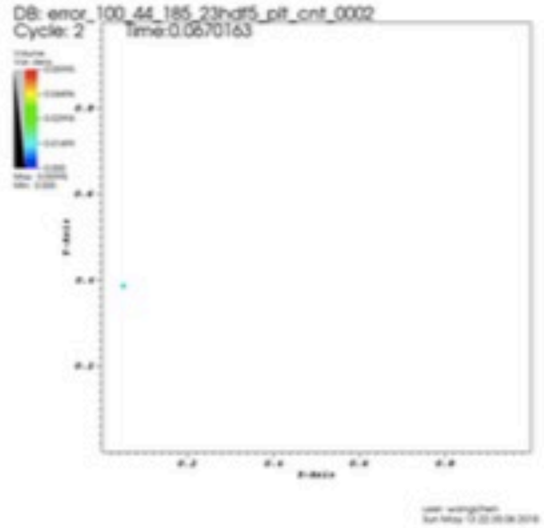
TABLE II: The impact of bit flips in different locations on the value 1.0 in single-precision floating point.

| Flipped bit | Value | Deviation |
|---|---|---|
| - | 1.0 | 0 |
| 31 | -1.0 | 2 |
| 30 | Infinity | Infinity |
| 29 | 5.421011E-20 | $\approx 1$ |
| 27 | 1.5258789E-5 | 0.99998474121 |
| 25 | 0.0625 | 0.9375 |
| 22 | 1.5 | 0.5 |
| 20 | 1.25 | 0.25 |
| 18 | 1.03125 | 0.03125 |
| 10 | 1.0004883 | 4.883e-4 |
| 5 | 1.0000038 | 3.8e-6 |
| 0 | 1.0000001 | 1e-7 |

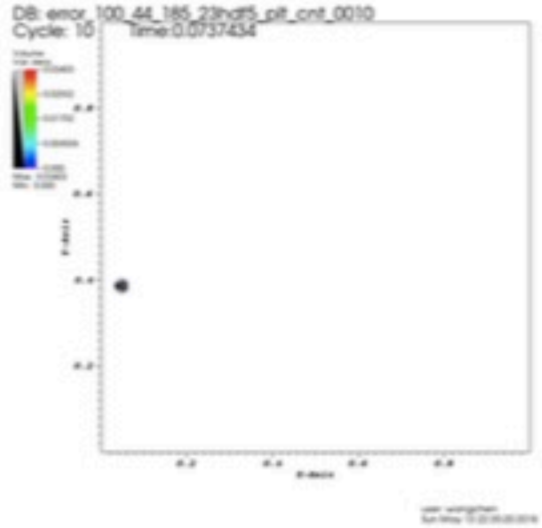
Silent data corruption (SDC)

TABLE III: Error positions

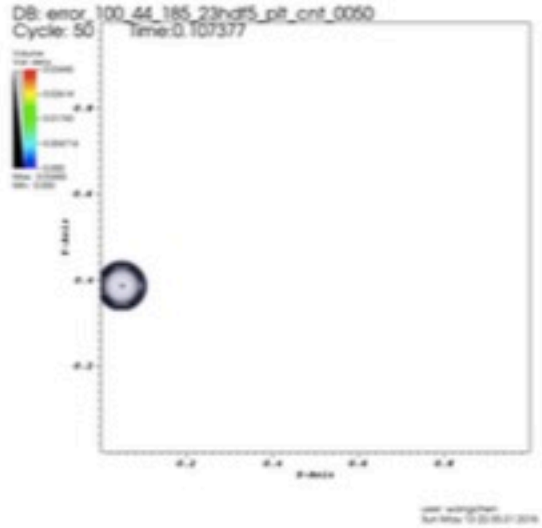
| App | Error bit position | Value changes |
| --- | --- | --- |
| Sod | 12 | From: 0.99999999999999989<br>To: 0.74999999999999989 |
| BrioWU | 11 | From: 1.0<br>To: 0.5 |
| BlastBS | 16 | From: 0.91620054602679901<br>To: 0.93182554602679901 |
| OrszagTang | 8 | From: 0.69874154473510708<br>To: 0.0027294591591215 12 |



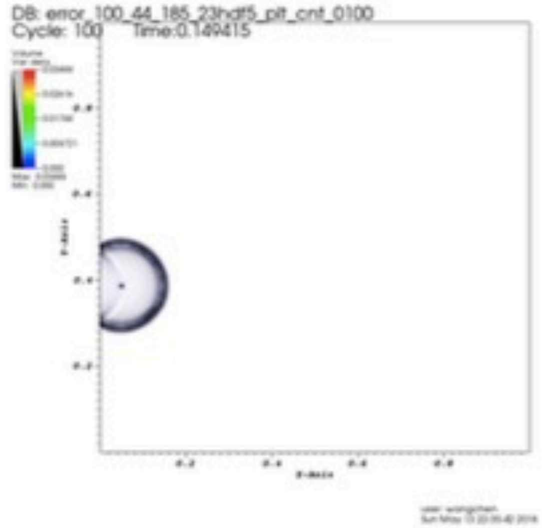

(a) Sod. 2 iterations after the error injection



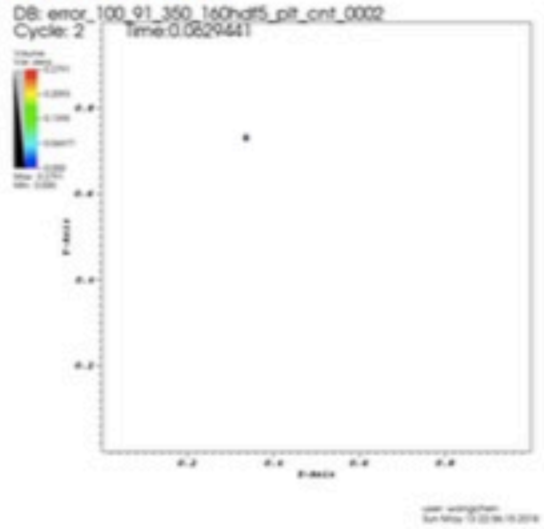

(b) Sod. 10 iterations after the error injection



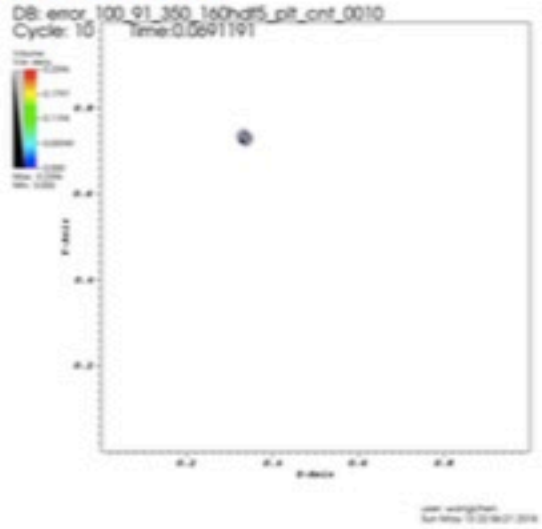

(c) Sod. 50 iterations after the error injection



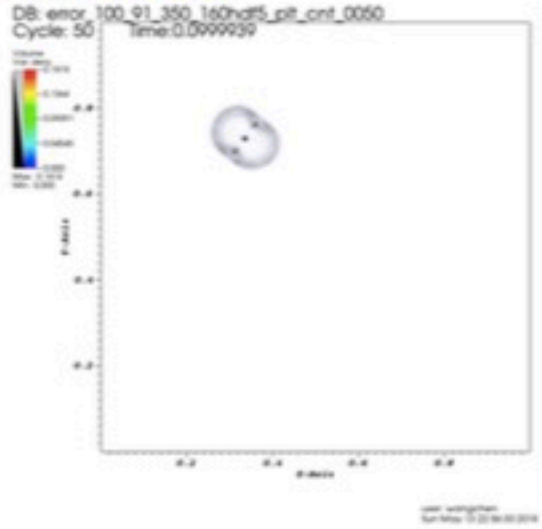

(d) Sod. 100 iterations after the error injection



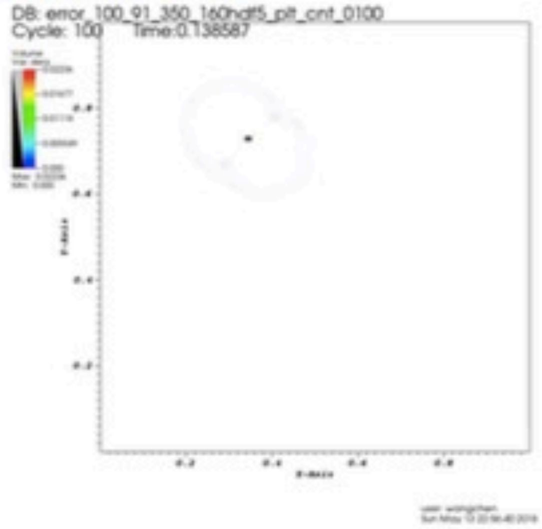

(e) BrioWu. 2 iterations after the error injection



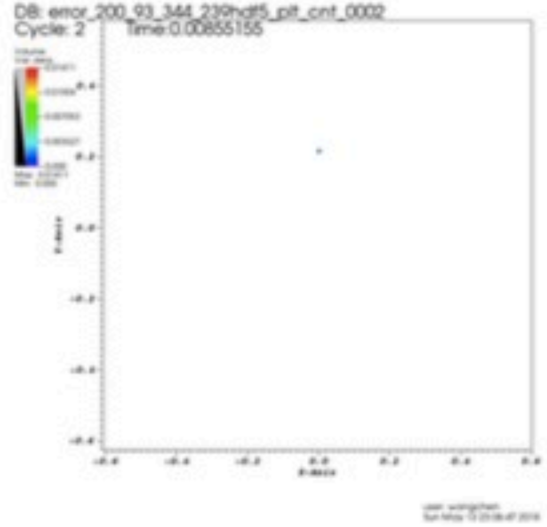

(f) BrioWu. 10 iterations after the error injection



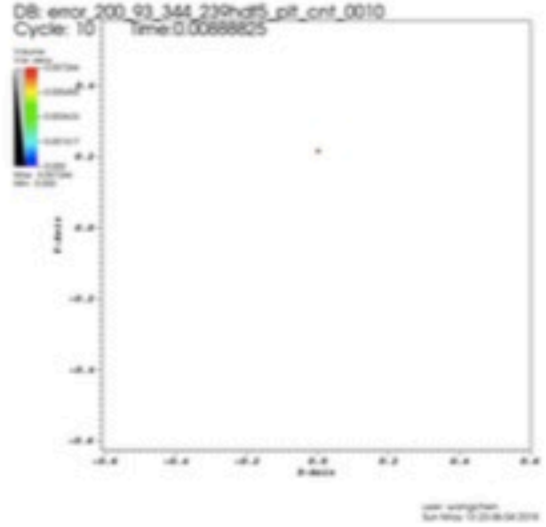

(g) BrioWu. 50 iterations after the error injection



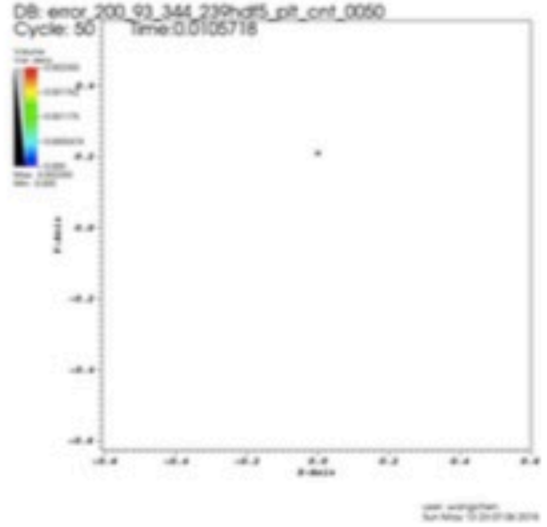

(h) BrioWu. 100 iterations after the error injection



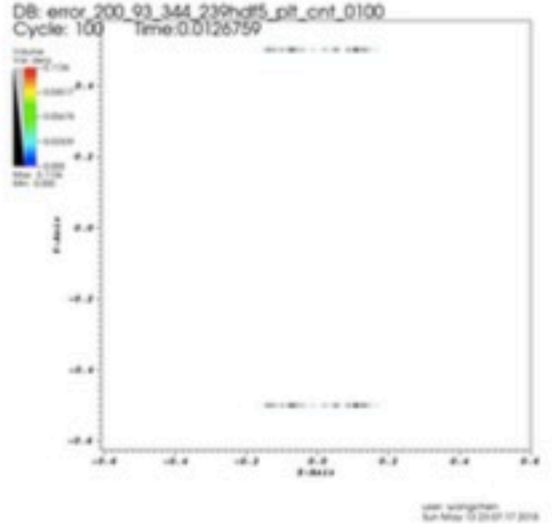

(i) BlastBS. 2 iterations after the error injection



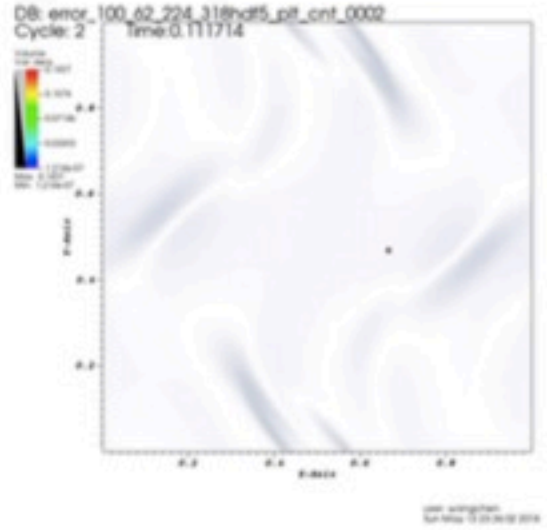

(j) BlastBS. 10 iterations after the error injection



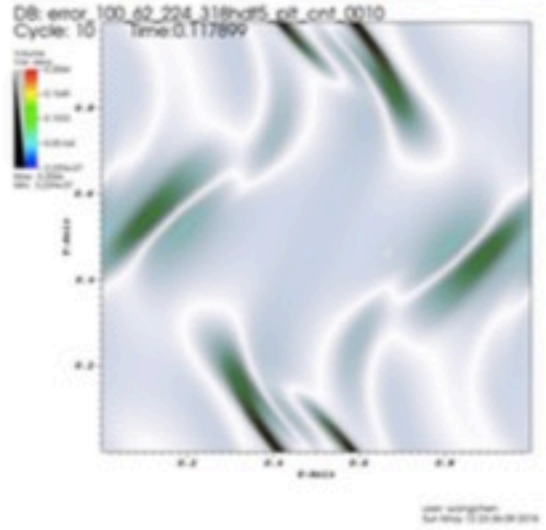

(k) BlastBS. 50 iterations after the error injection



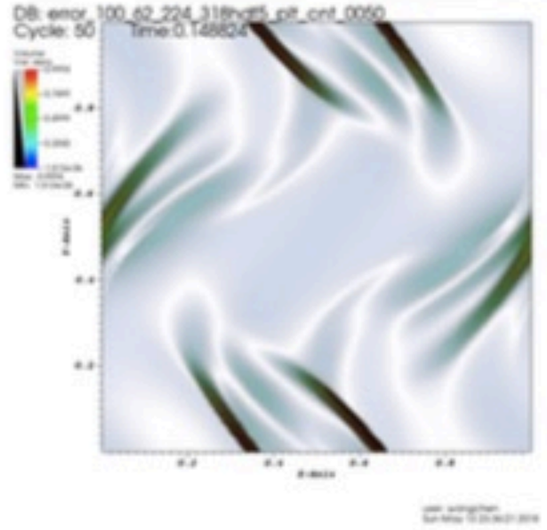

(l) BlastBS. 100 iterations after the error injection



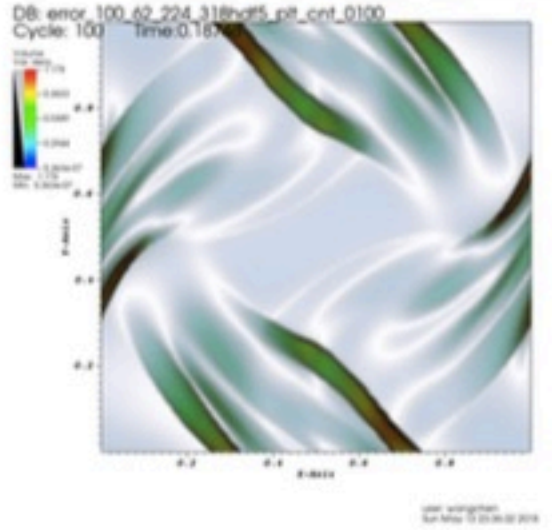

(m) OrszagTang. 2 iterations after the error injection




(n) OrszagTang. 10 iterations after the error injection




(o) OrszagTang. 50 iterations after the error injection




(p) OrszagTang. 100 iterations after the error injection

# Some movies

# Impact of SDC

$$I = \frac{\mathrm{sum}(\mathrm{abs}(v_{correct} - v_{corrupted}))}{\mathrm{sum}(v_{correct})}$$

TABLE IV: The impact of bit flips in different locations on the final result of applications. Errors in the top bits are significantly worse than errors in lower bits.

| App | bits 1-20 | bits 21-40 | bits 41-63 |
|---|---|---|---|
| Sedov | 16.62% (59187.704) | 0.36% (5.13e-4) | 0.36% (5.13e-4) |
| Sod | 20.43% (0.0333) | 0.05% (7.7391e-8) | 0.05% (7.491e-8) |
| BrioWu | nan | 0.04% (2.443e-6) | 0.04% (2.443e-6) |
| BlastBS | nan | 0.20% (2.685e-5) | 0.20% (2.685e-5) |
| DMReflection | 0.55% (0.1763) | 0.10% (4.137e-3) | 0.10% (4.137e-3) |
| OrszagTang | nan | 0.20% (4.855e-5) | 0.20% (4.855e-5) |

# Related Work

- Two Categories:
  - Temporal detector
  - Spatial detector

- Prediction Method:
  - Curve fitting
  - Machine learning

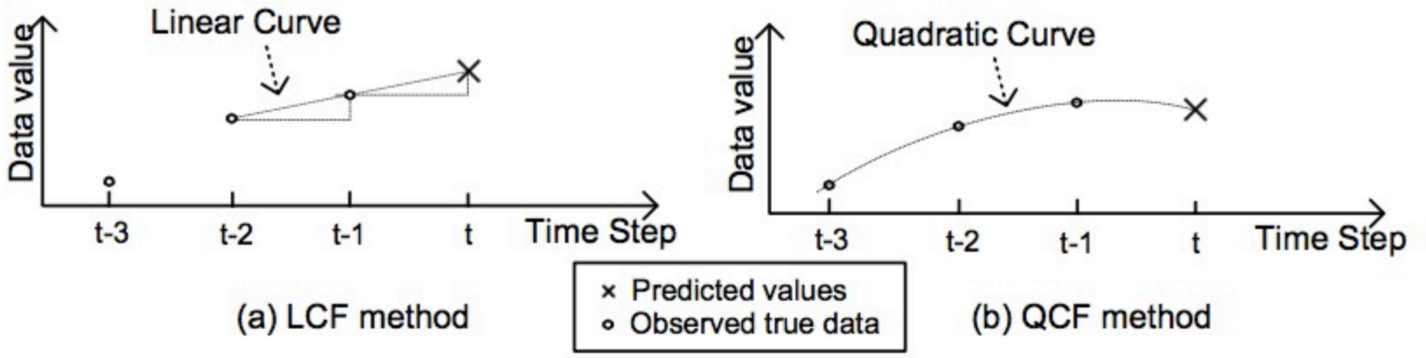



Fig. 10: Illustration of LCF and QCF

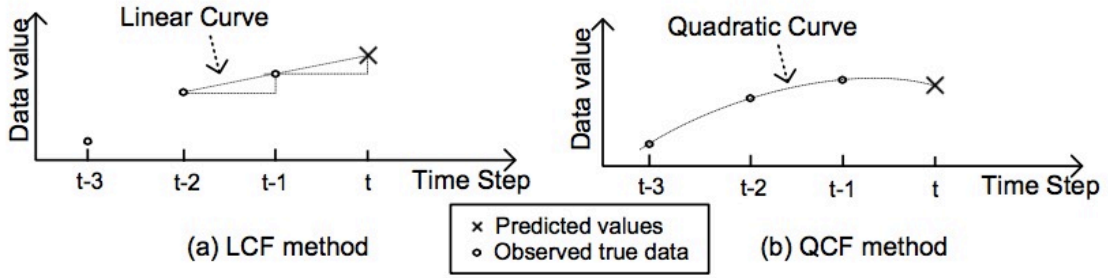
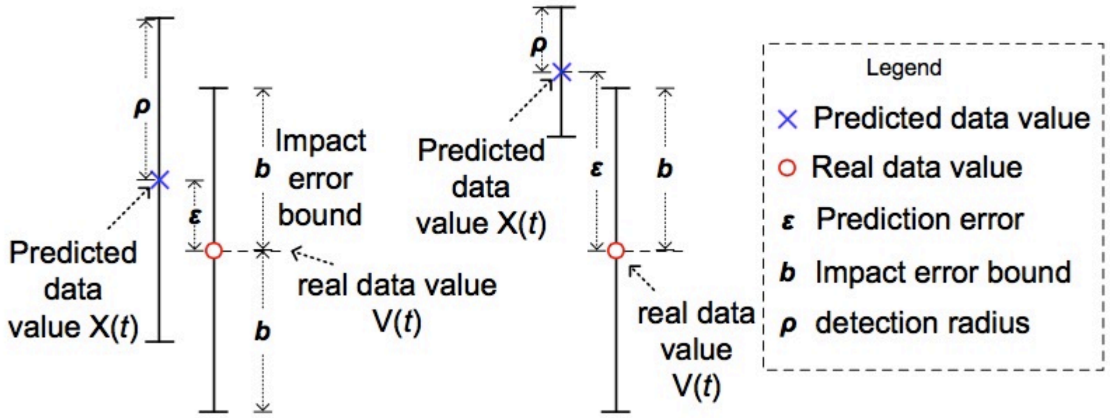


Fig. 10: Illustration of LCF and QCF



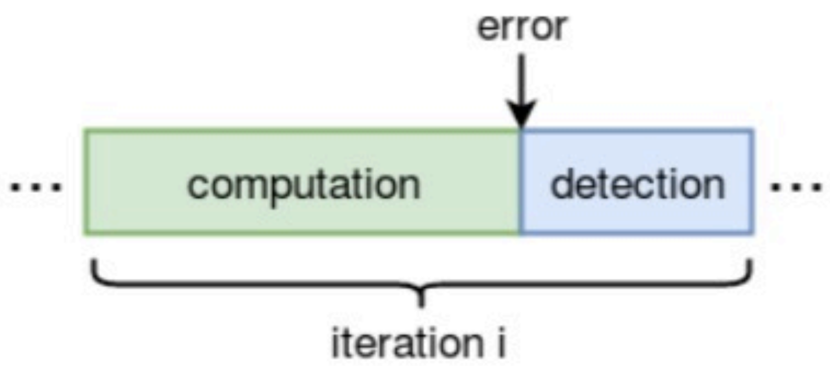
(a) valid prediction ($\varepsilon<b$) (b) invalid prediction ($\varepsilon\geq b$)

Fig. 9: Detection with Valid/Invalid Prediction

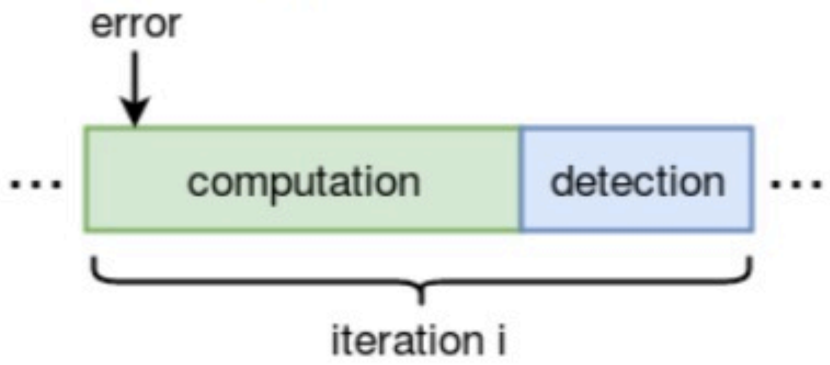
# State-of-the-art: adaptive impact driven detector (AID)

# Limitations of current work

- High memory footprint (4x for AID)
- High overhead
- Need to run the detector at ever iteration
- Recall & False positive
- Unpractical hypothesis




(a) An error occurs immediately before a detector is run. This common assumption simplifies error detection.




(b) An error occurs part-way through the computation within a single iteration. The application may smooth the error to some extent, making detection more difficult.

# System Overview

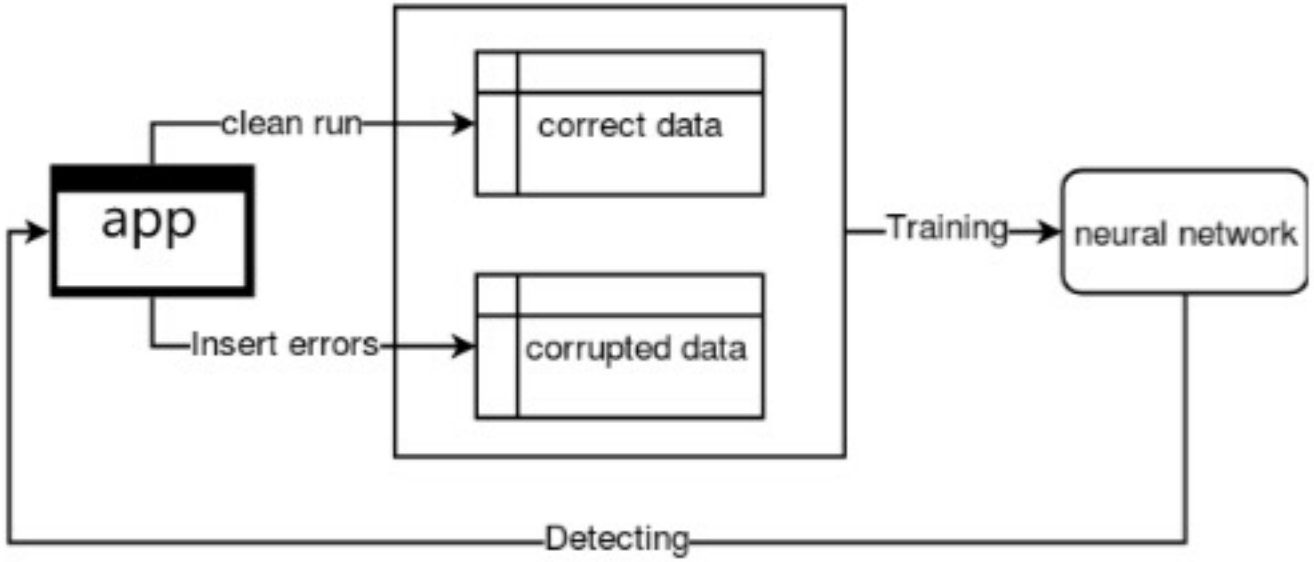



Fig. 3: System workflow: The application is used to generate training samples that are either clean or have artificial errors injected. A neural net is then trained on this dataset, and can then be used to detect errors in the application.

Our
Methods

# Neural network architecture

```
self.conv = nn.Sequential(
    nn.Conv2d(len(variables), 64, 5, stride=1),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(3, stride=1),

    nn.Conv2d(64, 64, 3, stride=1),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(3, stride=1),

    nn.Conv2d(64, 96, 3, stride=1),
    nn.BatchNorm2d(96),
    nn.ReLU(),
    nn.Conv2d(96, 96, 3, stride=2),
    nn.BatchNorm2d(96),
    nn.ReLU(),

    nn.Conv2d(96, 64, 3, stride=2),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.Conv2d(64, 32, 3, stride=1),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d(3, stride=1)
)
```

```
self.fc = nn.Sequential(
    nn.Linear(in_features=conv_output_size,
              out_features=512, bias=True),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(512, 1),
    nn.Sigmoid()
)
```

Our Methods

**Collect training/testing data**

1. Clean data

   we run each application for 1000 iterations with 10 different cases (initial conditions). We output variables we want to protect at every 5 iterations. The mesh size is set to 480 × 480, and we split it into 60 × 60 windows with the overlapping of 20

2. Corrupted data
   - Randomly flip a bit at random positions.
   - Restart from a corrupted dataset.
3. K-propagation dataset
   - Restart from a corrupted and save the corrupted data of following k iterations

**Split data into blocks**

- Allow concurrent detection

# Experimental Setup

- Blue Waters
  - Each compute node has 2 AMD 6276 Interlagos CPUs and 64GM of memory
  - Used to collect the training/testing dataset
- Nvidia DGX-1
  - 8 Tesla P100 GPUs
  - Used to train and test the detector

# Metrics

- Recall: the number of errors detected over the number of total errors
- False positive (FP): a time step is considered false positive if the detector reported an error when no error is present.

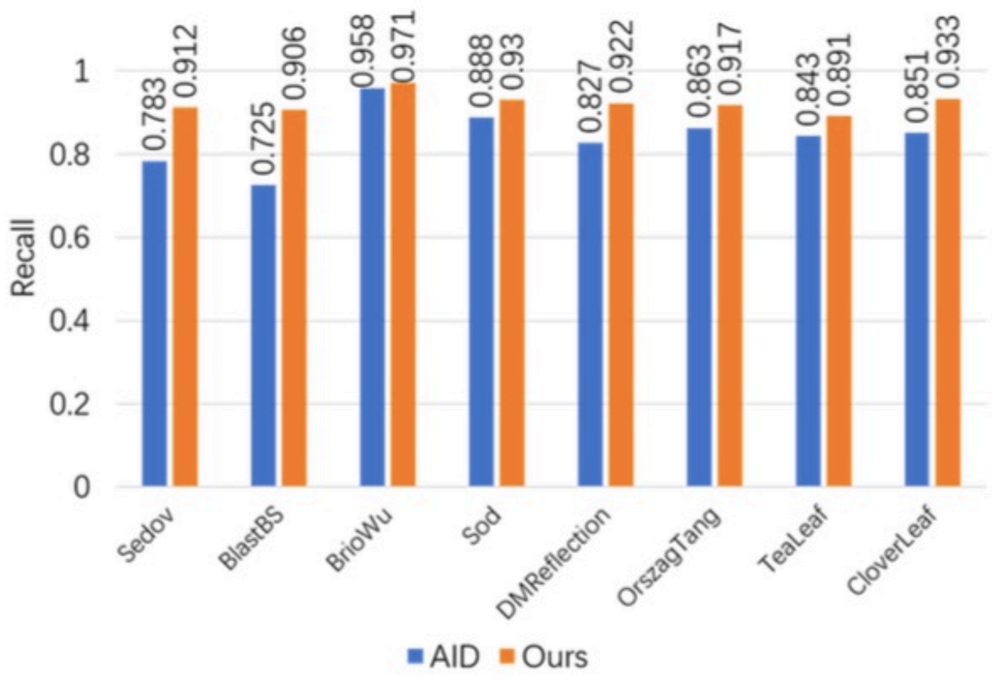



Fig. 4: Recall comparison with AID. Our detector achieves higher recall, outperforming AID.

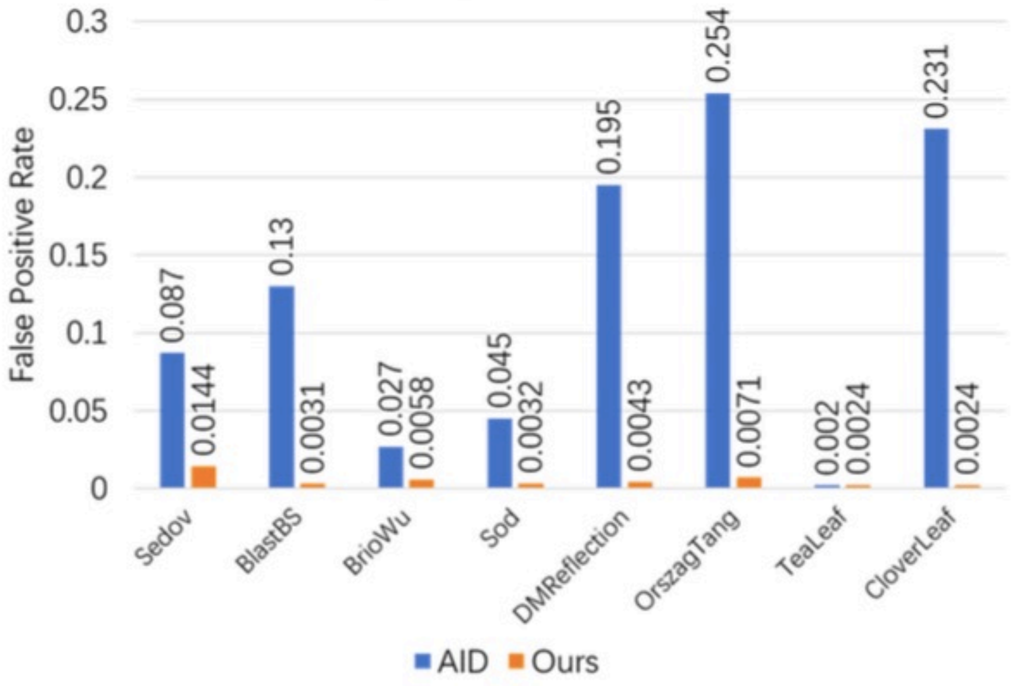
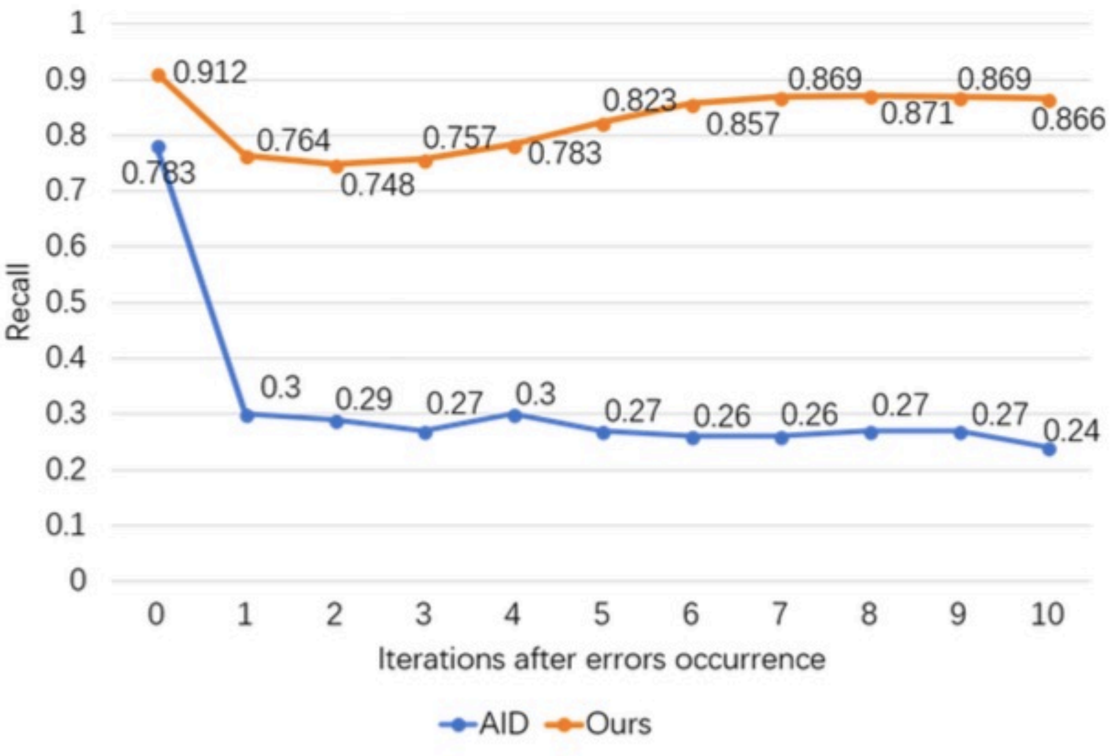
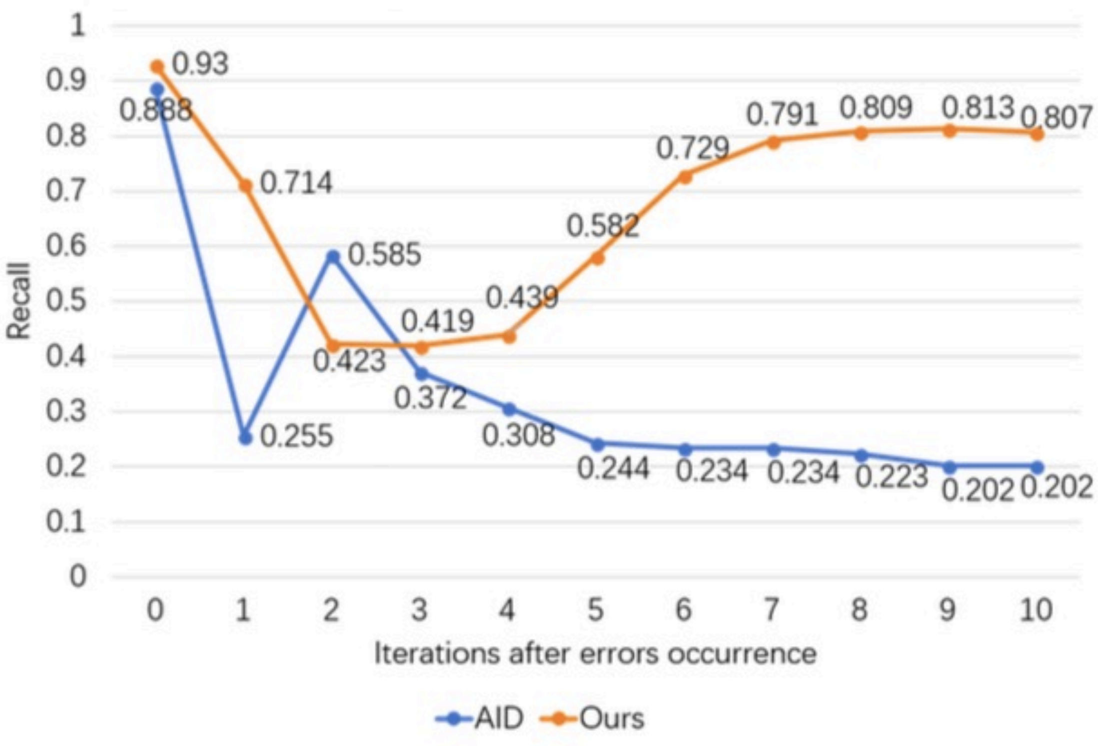



Fig. 5: False positive comparison with AID. Our detector has a significantly lower false positive rate.

# Comparison




(a) Sedov




(b) Sod

# Detect after multiple iterations

# Trained with k-propagation dataset

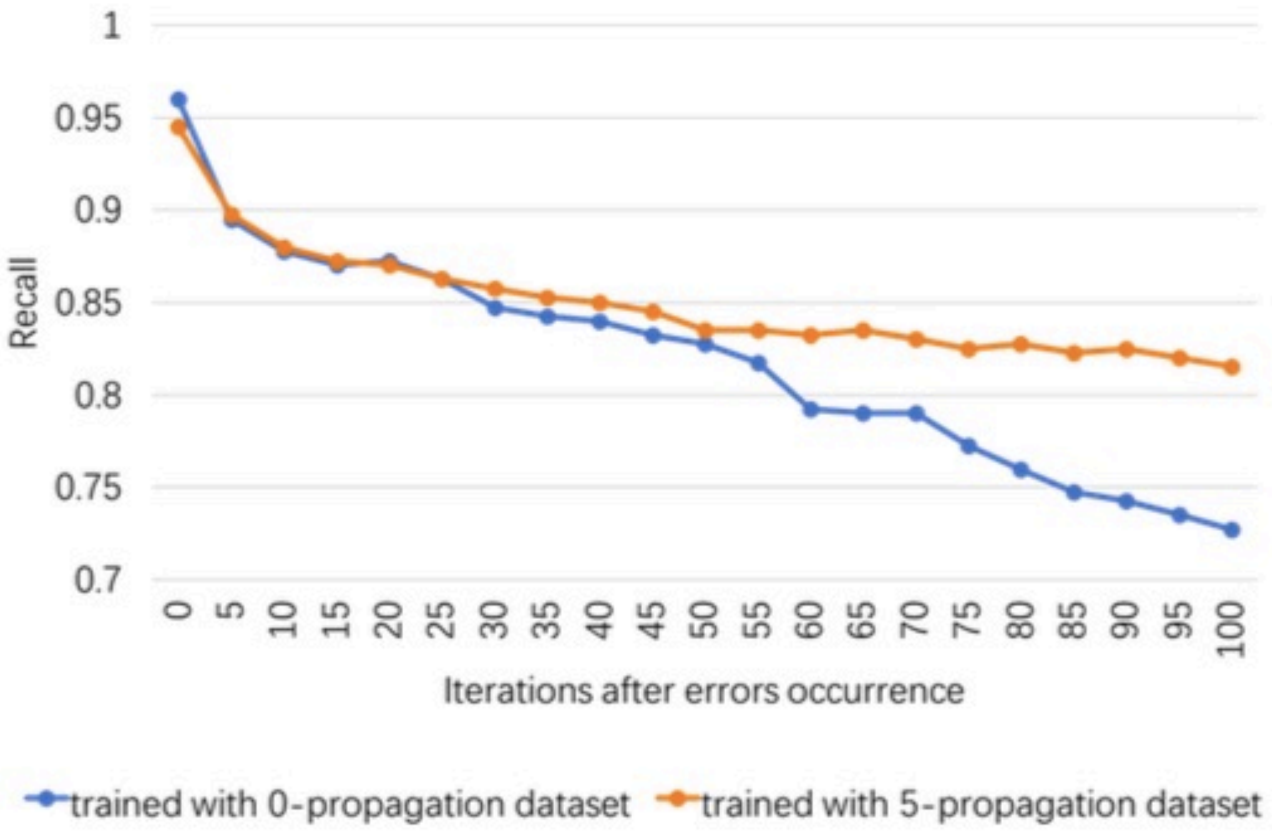



Fig. 7: Detecting errors in heat diffusion program with the neural network trained with 0- and 5-propagation dataset

# Future Work

- Checking whether errors that are not detected by our detector are smoothed over following iterations and do not corrupt the final result.
- How about detecting SDC in compressed (lossy) data ?
- Reduce overhead (0.011x on GPU and 2.951x on CPU)

# Q & A

# Thanks