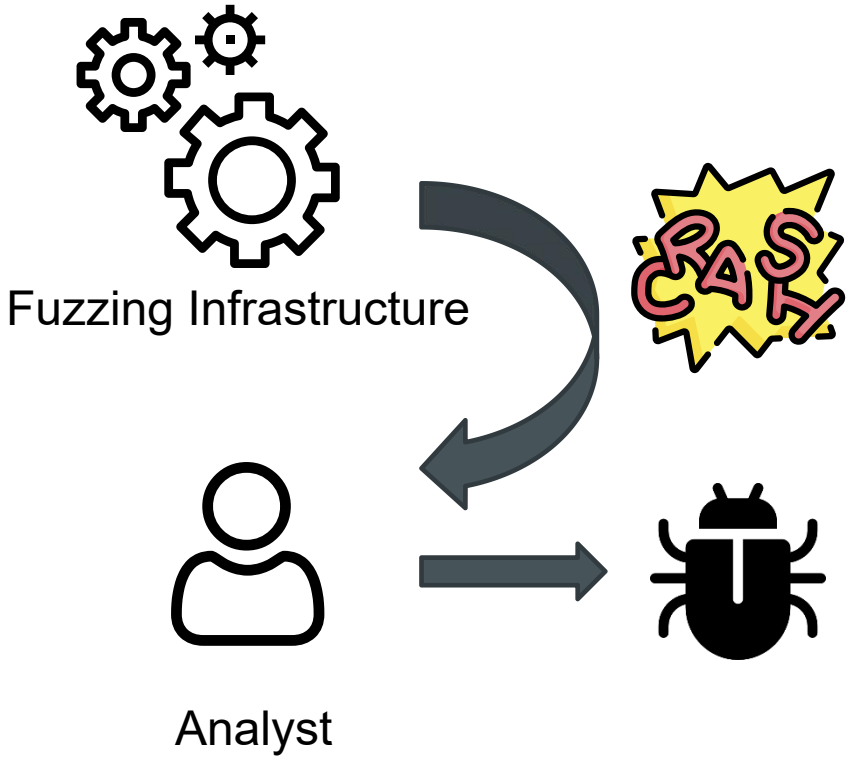
Reducing Test Cases with Attention Mechanism of Neural Networks

Xing Zhang, ***Jiongyi Chen***, Chao Feng, Ruilin Li, Yunfei Su, Bin Zhang, Jing Lei, and Chaojing Tang

National University of Defense Technology

What is Test Case Reduction?

- **Fuzzing is powerful**
  - As of February 2021, ClusterFuzz [1] has found ~29,000 bugs in Google and 26,000+ bugs in over 400 open source projects
- **Crashing inputs contains redundance**
  - A heavy burden for subsequent manual analysis
- **Test case reduction**
  - Minimizing crashing inputs by removing irrelevant portion and preserving failure-inducing portion



[1] ClusterFuzz https://github.com/google/clusterfuzz

Prior Efforts

Random Reduction

- Try to remove input blocks with different size [2][3]
  - Does not work when input blocks are inter-dependent

Rule-based

- Pre-define structures of inputs[4]
- Information flow tracking: specify the crash point and trace back to the input [5][6]
  - Inaccurate taint source: crash point != root cause point
  - Inaccurate propagation: control flow dependence

[2] C. Artho, “Iterative delta debugging,” International Journal on Software Tools for Technology Transfer, vol. 13, no. 3, pp. 223–246, 2011.
[3] A. Zeller and R. Hildebrandt, “Simplifying and isolating failure-inducing input,” IEEE Transactions on Software Engineering, vol. 28, no. 2, pp. 183–200, 2002.
[4] G. Misherghi and Z. Su, “HDD: hierarchical delta debugging”, in 28th International Conference on Software Engineering (ICSE 2006). ACM, pp. 142-151
[5] M. Carbin and M. C. Rinard, “Automatically identifying critical input regions and code in applications,” in Proceedings of the 19th international symposium on Software testing and analysis, 2010, pp. 37–48.
[6] J. Clause and A. Orso, “Penumbra: automatically identifying failure-relevant inputs using dynamic tainting,” in Proceedings of the eighteenth international symposium on Software testing and analysis. ACM, 2009, pp. 249–260.

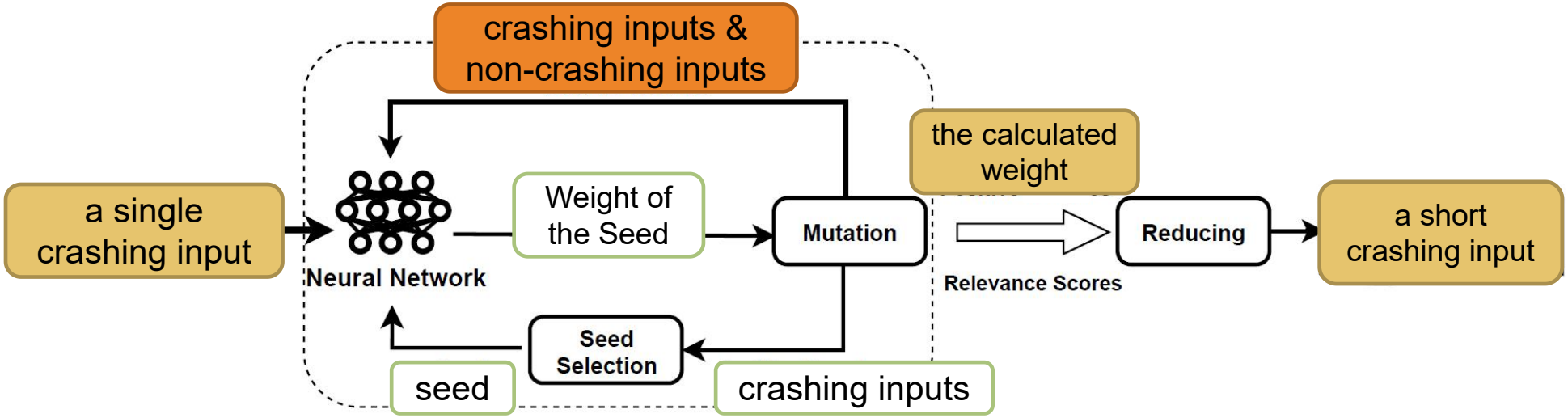
Our Insights

- Test case reduction is to determine a subset of input that contributes to the crash
- Leverage the **neural network** to denote:
  - **Significant & insignificant** input bytes (to the crash)
- Advantages
  - With **arithmetic expressions of math (rather than logical expressions of programs)**, the neural network flexibly approximates the program's computation
  - The interpretability of neural networks can give **contributions (i.e., weights) of each input byte**

Challenges

- How to construct a “good” dataset?
  - How to **automatically** generate samples and label them?
  - Training samples should be as **scattered** as possible in the sample space
- What is a suitable network structure?
  - Program inputs are **one-dimensional long** vectors
  - Existing network structures can only deal with short sequences

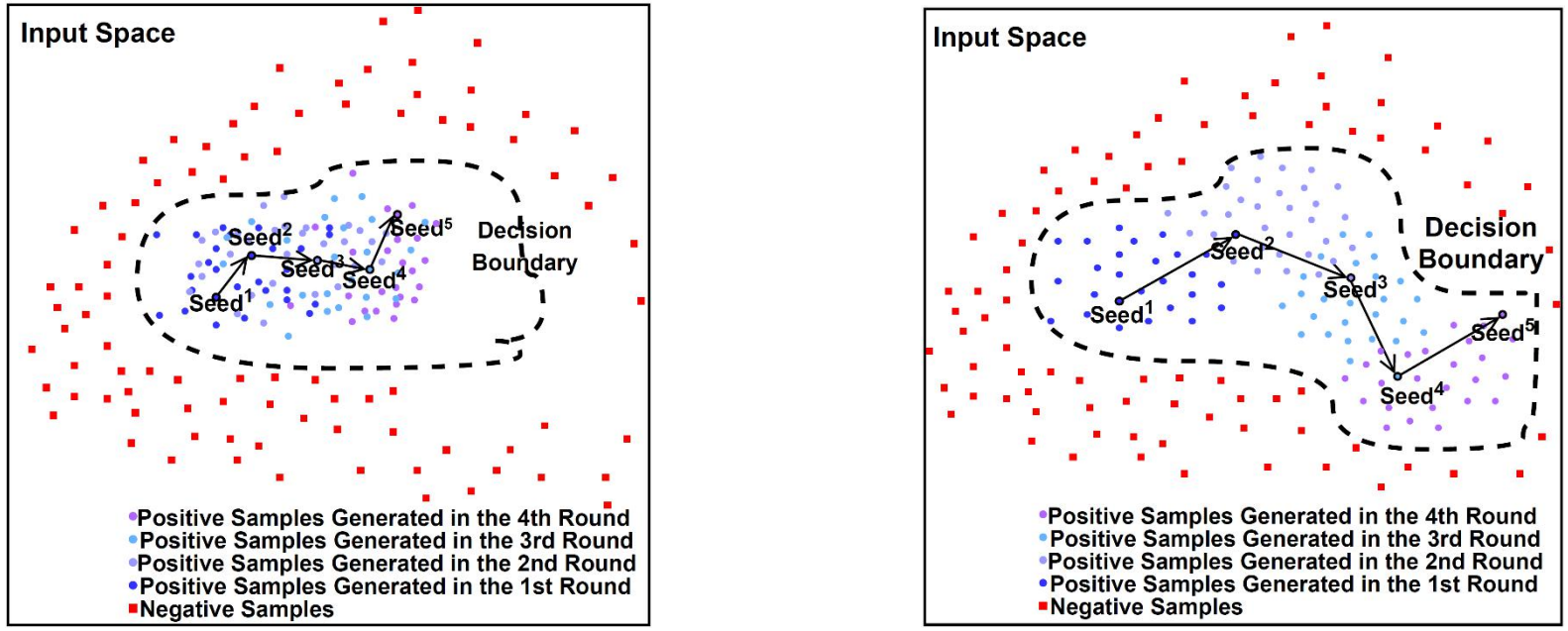
Overview of Our Approach



- **Dataset augmentation** is coupled with **network training**
- I/O of the neural network:
  - Feed the neural network with **crashing inputs & non-crashing inputs**
  - Give labels by **executing the program and observing if it "crashes" or "not"**
- The interpretability is leveraged to produce **weights of input bytes**

Dataset Augmentation

- Mutation:
  - Assign random mutation on a subset of input bytes
  - Generate **a set of crashing inputs** and **a set of non-crashing inputs**
- Seed Selection:
  - Choose the **most different one** from the set of crashing inputs



Distribution of dataset generated by existing approach (left) v.s. our approach (right)

Attention Mechanism for Interpretability

- Attention mechanism is utilized to extract the input weights that denote the contribution to the crash
- Transition equation with *softmax*:

$$\vec{\alpha} = \boldsymbol{softmax}(\boldsymbol{g}(\vec{\boldsymbol{x}}; \vec{\boldsymbol{\theta}}))$$

$$\vec{\boldsymbol{v}} = \boldsymbol{\alpha_1 x_1} + \boldsymbol{\alpha_2 x_2} + \ldots + \boldsymbol{\alpha_n x_n}$$

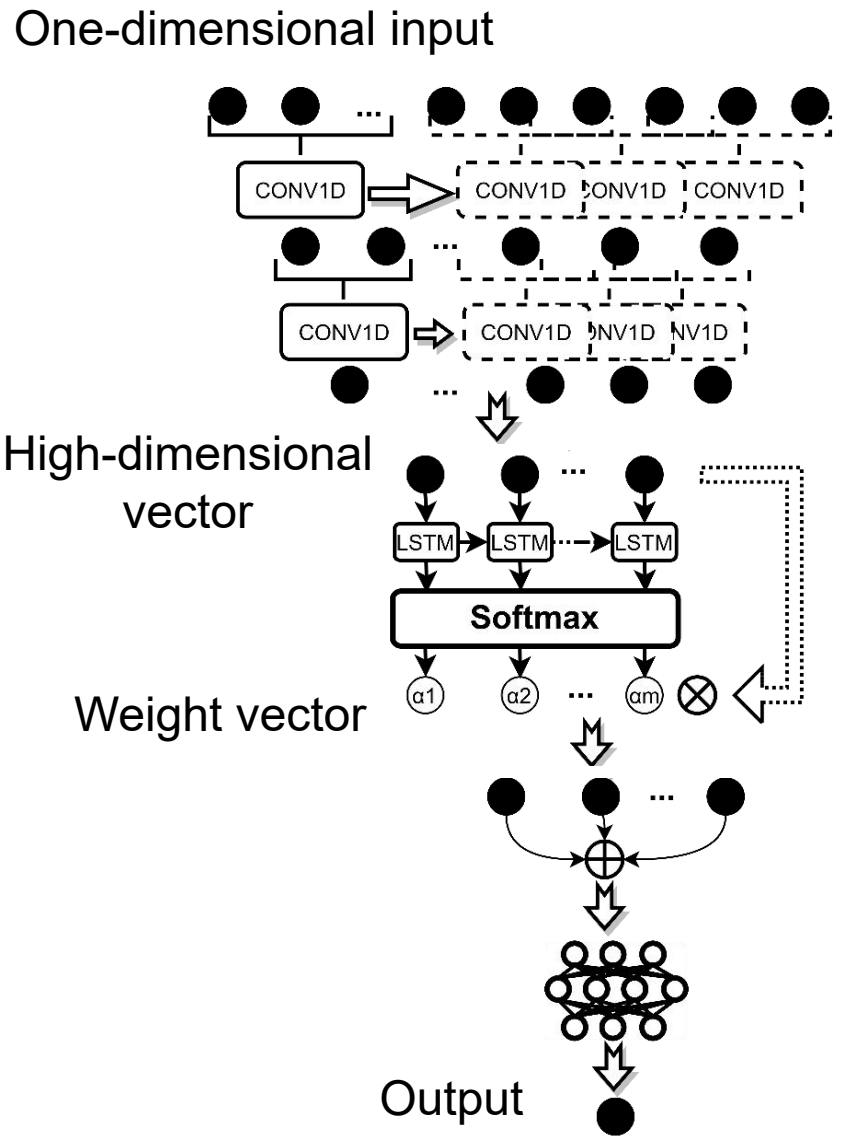
$$\boldsymbol{y} = \boldsymbol{f}(\vec{\boldsymbol{v}}; \vec{\boldsymbol{\theta}})$$

$$\boldsymbol{softmax}(\boldsymbol{x_i}) = \frac{\boldsymbol{e^{x_i}}}{\sum_j \boldsymbol{e^{x_j}}}$$

- Loss function: $L(\vec{\boldsymbol{\theta}}) = \sum_n \left| \boldsymbol{f}(\boldsymbol{softmax}\left(\boldsymbol{g}(\vec{\boldsymbol{x}}^{\boldsymbol{i}}, \vec{\boldsymbol{\theta}})\right) \odot \vec{\boldsymbol{x}}^{\boldsymbol{i}}; \vec{\boldsymbol{\theta}}) - \boldsymbol{y^i} \right|^2$

Network Structure & Weight Calculation

- **Network structure**: combine CNN with RNN
  - Encode one-dimensional long sequence as high-dimensional vectors
- **Weight calculation**: the length of weight vector < the length of input
  - Design backward weight allocation to compute the weight of each input node
  - Use multiple network instances to reduce imprecision



Reduction

- Based on the weights, re-assemble the significant input bytes

Input Bytes (w. Weights)	Crash?
 0.25, 0.05, 0.021, 0.02, 0.25, 0.05, 0.02, 0.25, 0.05	✓
 0.25, 0.05, 0.25, 0.05, 0.25, 0.05	✓
 0.25, 0.25, 0.25	✗

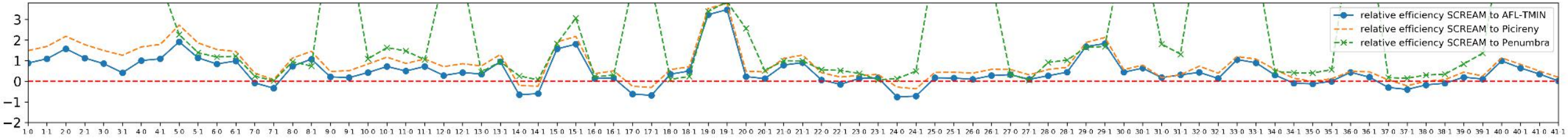
Evaluation Setup

- Implemented a prototype called *SCREAM*
- Evaluated with crashing inputs of 41 programs
  - Functionality of the programs: gaming, image processing, audio/video decoding, protocol parsing, etc.
  - Crashing inputs are produced with afl-fuzz
  - Randomly choose 2 crashing inputs per program

Results

- Average reduction rate of 75.4%, takes 29.8 minutes on average
- Compared with afl-tmin, Picireny, and Penumbra
- Efficiency of reduction: $E = \frac{size(I_{original}) - size(I_{result})}{T} \times \frac{size(I_{minimal})}{size(I_{result})}$

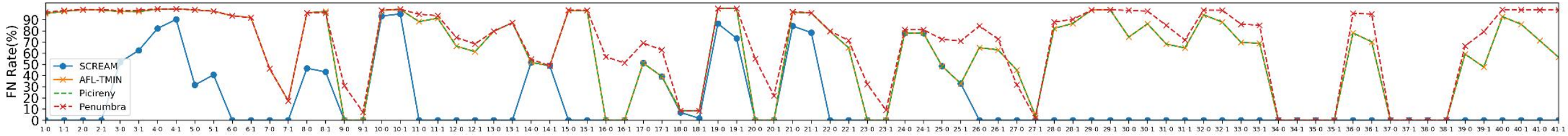
$log(R) = log(\frac{E_{scream}}{E_{AFL-TMIN}})$ $log(R) = log(\frac{E_{scream}}{E_{Picireny}})$ $log(R) = (\frac{E_{scream}}{E_{Penumbra}})$



Relative efficiency of SCREAM, afl-tmin, Picireny and Penumbra

Accuracy

- No false positives
- Average false negative rate is 17.0%
- With the help of SCREAM, 70.7% of the reduced inputs have reached ground truth



False negative rate of SCREAM, afl-tmin, Picireny and Penumbra

Strengths and Limitations

- Strengths:
  - Able to solve control flow complexity of crashing input in some cases:
    - multiple discontinuous input blocks that must be reduced at the same time
    - input blocks with specific format, e.g., the format of IP address
    - a field that specifies the minimal input length
    - etc.
- Limitations:
  - Handling complex arithmetic operations like checksum-like functions is beyond the expressiveness of neural networks

Thank you!

- The project is available at Github: https://github.com/zxhree/SCREAM
- Contact information: jiongyi_chen@126.com