Work-in-Progress: Accuracy-Area Efficient Online Fault Detection for Robust Neural Network Software-Embedded Microcontrollers

Juneseo Chang
Department of Computer Science and Engineering
Seoul National University
Seoul, Republic of Korea
jschang0215@snu.ac.kr

Sejong Oh
NVIDIA Corporation
Santa Clara, CA, USA
Sejongo@nvidia.com

Daejin Park
School of Electronics Engineering
Kyungpook National University
Daegu, Republic of Korea
boltanut@knu.ac.kr

***Abstract*—Detecting transient faults in safety-critical neural network (NN) applications operated on embedded systems has become a concern, but it is challenging to achieve high accuracy because of the open context problem and resource constraints. This study proposes an accuracy-area efficient, data-analysis-based online soft errors (SEs) and control flow errors (CFEs) detection, applicable to any NN application with low overhead. We insert code for runtime monitoring data assertion, and the data are distributed to shallow or deep detection models selectively. The shallow detection model detects CFEs by verifying runtime signatures with values obtained from simulations, and detects SEs of data having constant values according to program input. SEs of other data are verified by a deep detection model using a sliding window one-class support vector machine. Fault injection experiments on an image classification NN showed that our detector has significant detection accuracy in fault conditions.**

***Index Terms*—Fault Tolerance, Embedded System, Neural Networks, Reliability**

I. Introduction

Today, the development of microcontroller (MCU) performance and lightweight neural network (NN) models enabled NN applications to be deployed on MCUs. Among them, some must tolerate faults or runtime errors, i.e., are safety-critical. For instance, vehicle steering controllers, medical diagnostics tools, and infrastructure controllers utilize NNs in safety-critical conditions. [1]

While there are various software runtime errors, we focus on control flow error (CFE) and soft error (SE). CFEs are errors when the instruction sequence is different from an error-free environment. [2] SEs are transient bit flips in memory that change data values on memory. Since a single SE or CFE causes fatal outputs in safety-critical applications, it is crucial to detect software faults immediately; thus, fault detection schemes should be employed. Among various detection methods, we focus on a data-analysis-based model that observes the target program's processing values online. The data-analysis-based model does not require complex hardware modification and is not algorithm-specific. Therefore, it can easily be applied to any target program and device. [3] Meanwhile, unlike non-NN, which process fixed inputs, NNs are given different inputs that are of the same label. Since there are almost infinite possible inputs for each label, we can not formally specify all possible variable values of the application. This characteristic of the NN application is called the open context problem, which makes detecting faults in NNs by a data-analysis-based model challenging.

In this study, we propose a data analysis-based, general-purpose, off-the-shelf CFE and SE online detector with low overhead, which satisfies the embedded system environment. We devise a selective CFE and SE detection algorithm which can be applied to any program without hardware modification. We utilize a simple image classification NN or fault injection experiments for evaluation.

This study was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1A6A1A03025109, 10%, NRF-2022R1I1A3069260, 10%), by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00944, 50%, No. 2022-0-00816, 20%, No. 2022-0-01170, 10%).

II. Implementation

A. Code Insertion Process

To monitor the CFEs and SEs at runtime, processing basic block index or memory values should be asserted to the detector; therefore, a function for data assertion should be inserted. Since user-driven code insertion is cumbersome, and code insertion in a high-level language could lose language portability, we developed an LLVM IR parser to insert code at the IR produced by the LLVM frontend. For the basic block index assertion, our tool assigns a unique index for each basic block and asserts it at the end of every basic block. For SE detection, because every memory is at risk of transient bit flips, the memory value is asserted at the end of every *store* instruction.

B. Detector Architecture

The proposed detector determines errors based on control flows and memory values obtained from correct executions. Various control flows and memory values are obtained by performing multiple executions of the target program with varying program input. To identify control flow, we obtain the sequence of basic block indexes from correct executions. For memory values, we classify them as *constant* or *nonconstant* data if it has the same value for every simulation for each control flow. Examples of *constant* data are weights and bias, while *nonconstant* data are values stored in neurons. At runtime, the basic block index and memory values are transmitted

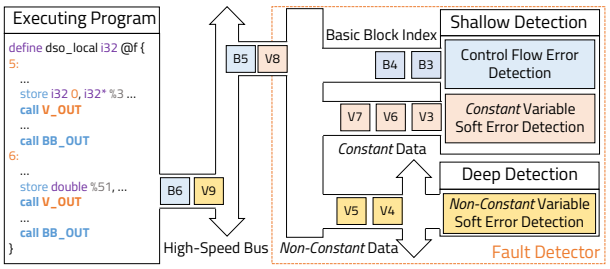

Fig. 1. Proposed detector architecture

to the detector and analyzed with selective detection models (Figure 1). The basic block index and *constant* data are sent to the shallow detection model, where basic block indexes are compared with the correct control flow obtained from the correct executions. [2] From this operation, we can get candidates of the target program's expected control flows. The *constant* are compared with data values obtained from correct executions that have control flow of the expected control flows.

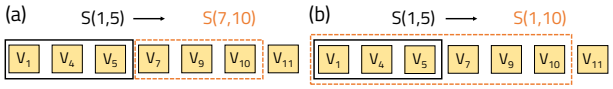

Fig. 2. Sliding window OCSVM model (a) Window sliding when the first window is labeled correct and (b) erroneous

The *nonconstant* data vary according to program input, so it is impossible to detect SEs for *nonconstant* data by comparing them with values obtained from correct program executions. To verify whether the asserted *nonconstant* data have acceptable values compared to the correct values, our deep detection model uses a supervised learning model trained from correct execution data. Since the deep detection model should detect abnormal values online with low false-positive and false-negative rates, we developed a specific sliding window OCSVM model. The proposed sliding window scheme changes the window size according to each window's OCSVM results. For instance, if the OCSVM for the first window labels the data in the window as correct, it slides to the next window. (Figure 2-(a)) However, if the OCSVM labels the first window as erroneous, the window size increases. (Figure 2-(b)). If the window size is larger than the maximum window size, the detector labels the program execution as erroneous. We used the OCSVM with radial basis function with scale kernel coefficient and the optimal ν selected from the F1-score look-up table.

C. Test Program and Fault Injection

Our test program is a simple image classification application that classifies images into a single label. The NN has 1 hidden layer with 8 nodes. We used 5,000 MNIST images to obtain correct execution data for shallow detection and train the deep detection model, while 500 images were used for evaluation. To simulate a fault situation, we processed the asserted data from the test program through the fault injection model and passed the data to the detector. Our fault injection model takes error rate, number of bits to be flipped, and probability distribution of bit flip location as parameters. [4]

III. Experimental Results

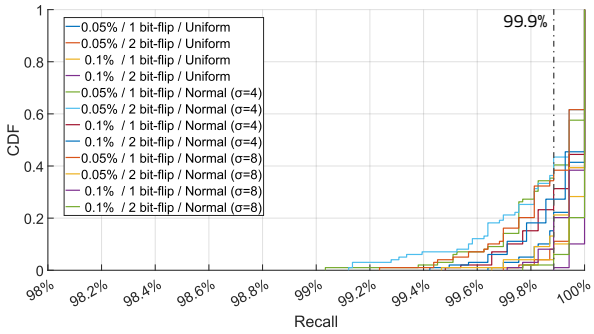

Fig. 3. The CDF of recall for SE detection

We performed 1.2 million experiments with 0.05% and 0.1% error rate, 1 and 2 bit flip, uniform and normal distribution with std 4 and 8. We defined false negative as failing to detect an error of erroneous execution. [4] The shallow detection model's CFE detection recall and precision were 100%. Figure 3 shows the CDF of the recall for the SE detection, where our detector's recall reached above 99.9% in most cases. The SE detection precision was 99.21%. The increased target program binary file size was 14.4%.

IV. Conclusion and Outlook

We designed a novel data analysis-based online CFE and SE detector for NN applications that can be applied to any target application with low overhead. We selectively analyzed processing data with the shallow and deep detection models. Experiments on an image classification model verified that our detector has significant recall and precision. We plan to work on implementing our model on widely used MCUs; for instance, the target program and shallow and deep detection models can be implemented on each core of a tri-core MCU, and the deep detection model can be handled by a callback to a high-performance system. Additionally, we plan to work on rollback operations, which can be performed by partial replacements on erroneous variable values.

References

[1] C.-H. Cheng, F. Diehl, G. Hinz, Y. Hamza, G. Nuehrenberg, M. Rickert, H. Ruess, and M. Truong-Le, "Neural networks for safety-critical applications — challenges, experiments and perspectives," in *Design, Automation Test in Europe Conference Exhibition*, 2018, pp. 1005–1006.

[2] K. Choi, D. Park, and J. Cho, "Sscfm: Separate signature-based control flow error monitoring for multi-threaded and multi-core environments," *Electronics*, vol. 8, no. 2, 2019.

[3] S. Di and F. Cappello, "Adaptive impact-driven detection of silent data corruption for hpc applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2809–2823, 2016.

[4] G. Kestor, B. O. Mutlu, J. Manzano, O. Subasi, O. Unsal, and S. Krishnamoorthy, "Comparative analysis of soft-error detection strategies: A case study with iterative methods," in *15th ACM International Conference on Computing Frontiers*, 2018, p. 173–182.