Extract Useful Features for Detecting Transient Faults

Zhiqiang Sui, Zhefan Ye, Karthik Desingh

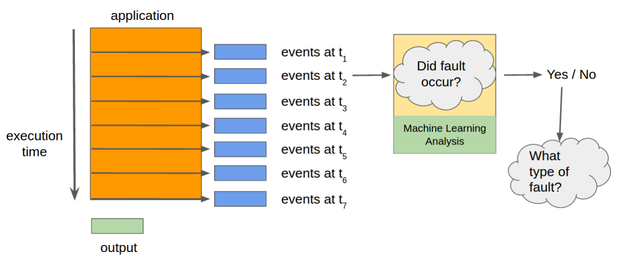

Figure 1: Overview

ABSTRACT

1. INTRODUCTION

In this report, we present a pipeline that can inject faults into programs and analyze how injected fault can alter program outcomes. To detect transient fault has been a research topic in robust system design for years. We utilize gemFI to study how faults are going to affect programs. Our contribution is two-fold: we implemented a fault inject pipeline, and provide analysis on how to predict fault and program outcome based on a machine learning approach. Figure 1 shows the overview of our approach.

2. RELATED WORK

3. APPROACH

In this section, we present our implementation for injecting faults, as well as analysis for predicting program outcome and fault types.

3.1 Pipeline

Figure 2 shows the pipeline.

3.2 Fault Injection

3.3 Feature Extraction

3.4 Machine Analysis

We use the random forest algorithm to predict program outcome and fault type. A random forest is essentially an ensemble of single decision trees, as illustrated in 4 [1]. It captures different models of the data, with each decision

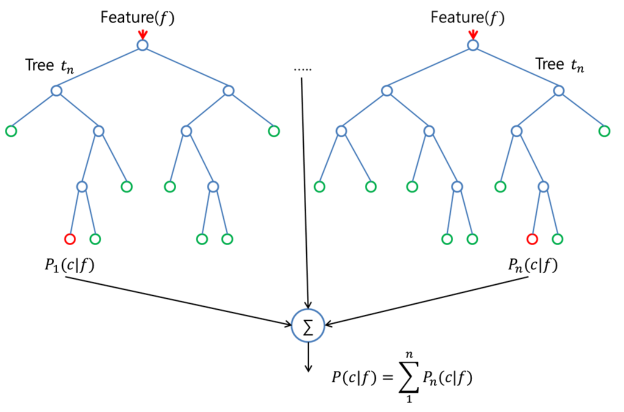

Figure 4:

tree representing a model, and allows us to analyze the importance of different features.

3.4.1 Training and Testing

We randomly select 60% of instances for training and 40% of instances for testing. Our dataset consists of 98,000 data instance.

4. EXPERIMENT

4.1 Metrics

we use Precision-Recall (PR) curve and F_1 score as our evaluation criteria. More precisely, we have

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{1}$$

where $precision = \frac{tp}{tp+fp}$, $recall = \frac{tp}{tp+fn}$, tp is the number true positive samples, fp is number of false positive samples, and fn is the number of false negative samples. For multi-class classification, we use confusion matrix to describe the performance of our classifier.

We design four experiment setups: 1) same input data with all meaningful features, 2) same sets of input data with handpicked subsets of features, 3) different sets of input data with all meaningful features, and 4) different sets of input data with handpicked subsets of features.

4.2 Same Input All Features

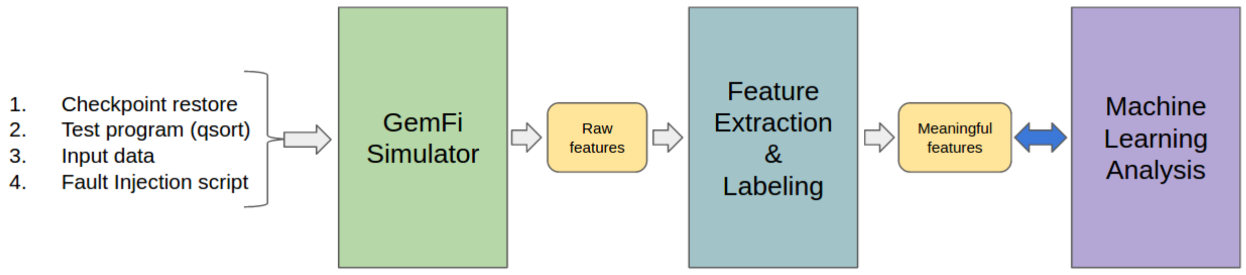

Figure 2:

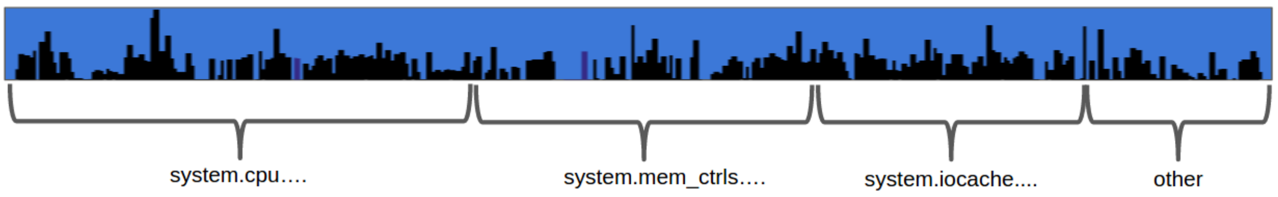

Figure 3:

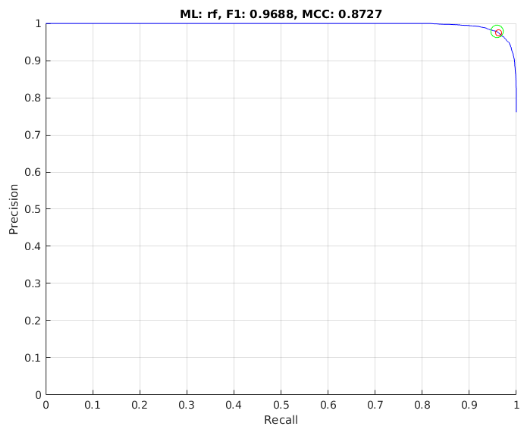

Figure 5:

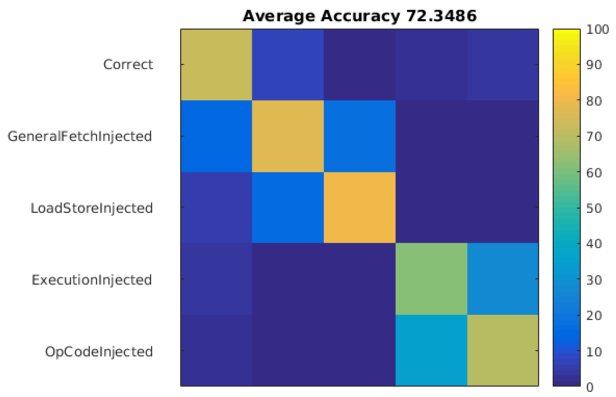

Figure 6:

(same input all features SIAF)

The random forest algorithm output the importance score of each feature based on its discriminative power. The importance feature ranking is depicted in Figure 7.

4.3 Same Input Different Features

(same input handpicked features SIHF)

4.4 Different Input All Features

(different input all features DIAF)

4.5 Different Input Different Features

Different Input Handpicked Features DIHF

5. DISCUSSION AND CONCLUSION

In the same input scenario, a few features alone can determine the results in same input condition. For same input the most important features are mainly describing the execution length. For d different input scenario: the features such as, âĂIJtype of functional units issuedâĂİ, âĂIJfetch instructionsâĂİ and âĂIJcache read and writeâĂİ, play important role in the prediction. Overall features based on âĂIJL2 cacheâĂİ events perform higher than other set of hand-picked features. However, when all the features are used, the performance of the random forest algorithm is high. We extract meaningful fault signatures from the architectural events that can be used to predict transient fault occurrence.

6. REFERENCES

[1] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1,

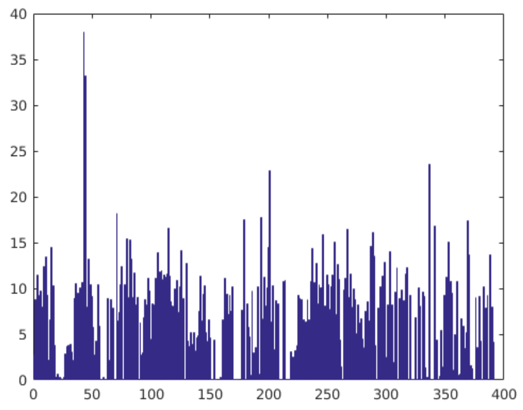

Figure 7:

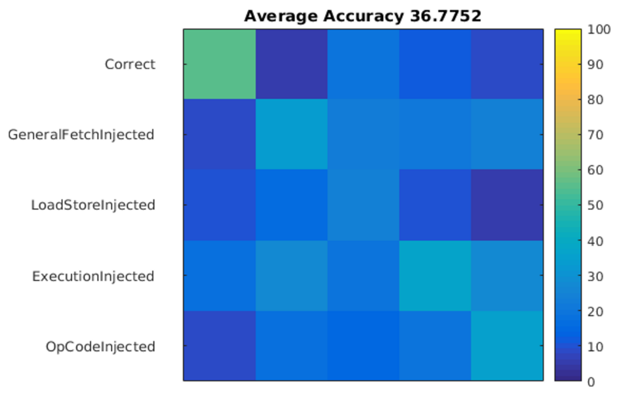

Figure 9:

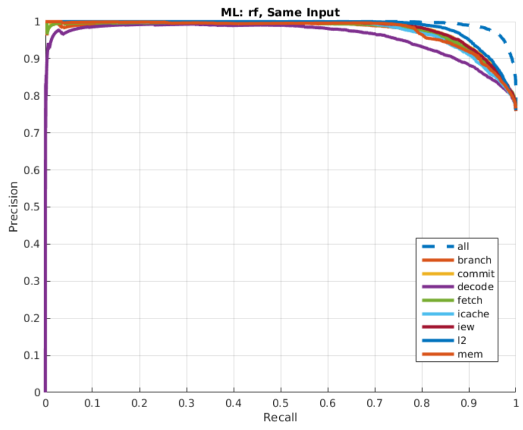

Figure 8:

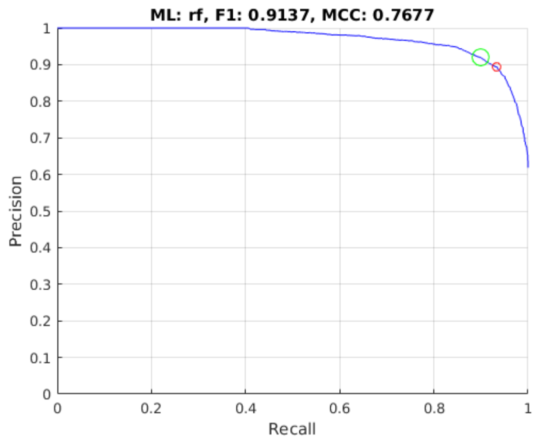

Figure 10:

pp. 5–32, 2001.

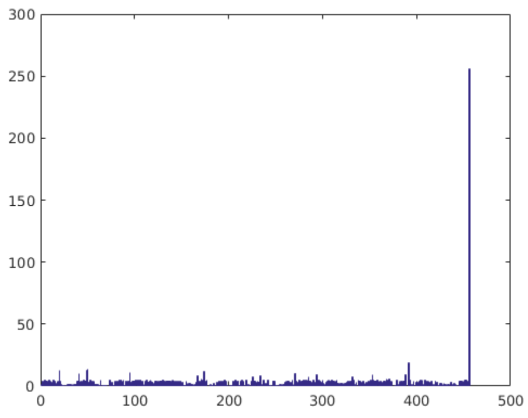

Figure 11:

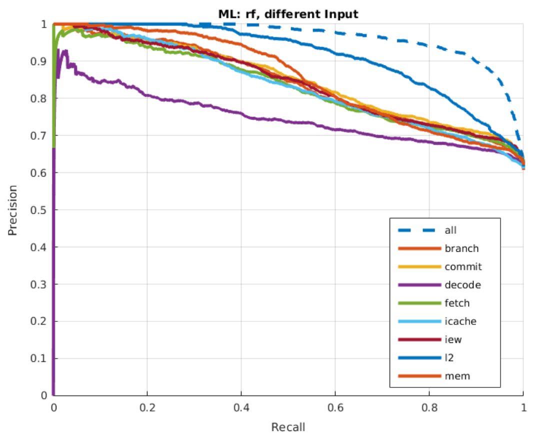
ML: rf, different Input
Precision
Recall
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1
all
branch
commit
decode
fetch
icache
iew
l2
mem

Figure 12: