FIBR-OSS: fault injection model for bug reports in open-source software

Sundos Abdulameer Alazawi[1], Mohammed Najm Al-Salam[2]
[1]Department of Computer Sciences, Mustansiriyah University, Iraq
[2]Department of Computer Engineering, University of Technology, Iraq

Article Info

Article history:

Received Jan 2, 2020
Revised Apr 4, 2020
Accepted Apr 18, 2020

Keywords:

Bug
FIBR-OSS
Injection techniques
Open-source software

ABSTRACT

For assessment of system dependability, fault injection techniques are used to expedite the presence of an error or failure in the system, which helps evaluate fault tolerance and system failure prediction. Defects classification and prediction is the principal significant advance in the trustworthiness evaluation of complex software systems such as open-source software since it can quickly be affected by the reliability of those systems, improves performance, and lessening the product cost. In this context, a new prototype of the fault injection model is presented, FIBR-OSS (Fault Injection for Bug Reports in Open-Source Software). FIBR-OSS can support developers to evaluate the system performance during phase's development for its dependability attributes such as reliability and system dependability means such as fault prediction or forecasting. FIBR-OSS is used for fault speed-up to test the system's failure prediction performance. Some machine learning techniques are implemented on bug reports produced existing by the bug tracking system as datasets for failure prediction techniques, some of those machine learning techniques are used in our approach.

Corresponding Author:

Sundos Abdulameer Alazawi,
Department of Computer Sciences,
Mustansiriyah University, Baghdad, Iraq.
Email: ss.aa.cs@uomustansiriyah.edu.iq

1. INTRODUCTION

The Society is now increasingly dependent on open-source systems, which have become an important part of daily communication and interaction factors in the world. Most open-source systems have a direct and significant impact on basic services such as telecommunications systems for commercial and government companies, transport, health, and many others. Due to the proliferation of those software and systems in our real world and the significant impact on them, it has become important and necessary to ensure the dependability of these systems and their ability to provide services efficiently and reliably even if allowed to pass some faults without compromising the reliability of the services provided by those systems [1, 2].

Errors and faults may take place in any software development phases and most software companies are focus on software dependability especially over those phases. Therefore, the key topical of each company is to define and correct/remove of errors and faults. For this, failure prediction and data mining techniques are implemented [3]. Software faults are more costly and consumption of time, the cost of detecting and repair faults represents one of the maximum costly software development actions [3, 4] for this, fault injection techniques are used to expedite the presence of an error and then decrease of cost of time for testing and evaluate of system behavior under development [5, 6].

Machine learning is a discipline that deals with the development of algorithms to make the computer systems exhibit intelligence in their behavior, its focus is to make systems learn from the training data and to behave intelligently when a new data set is given. Machine learning accomplishes this using complex relationship modeling among the data entries in the training set [3, 7]. This behavior of systems is predictable using machine learning, predicting system failures before they occur.

Through studying and analyzing the reviews and publications on the subject of this project in [8], we found that faults injection is a confirmed and significant method for assessment of software dependability. To inject faults, researchers and developers have created several new techniques that can be performed in hardware, software, or both. The subsections of software fault injection techniques are Data Errors Injection, Interface Injection, and Change Code Injection [2, 9, 10].

Software fault injection techniques can be implemented in many types of software and at various levels of abstraction within the software, such as Operating Systems Level compares the dependability for different Operating system calls, Interfaces Level by injected the faults at interface level for application and its libraries, and Distributed System Level based on hardware and message passing [2, 9]. The concept dependability is an ability to deliver service and avoid service failures that are more frequent and severe than of user can accept it [11].

Our approach is implemented on four open-source software Linux kernel, MySQL DBMS, Apache HTTPD web server, and Apache AXIS WS. In fact, we visualize our plan in which the behaviors of software collected during faults injection can be used in failure prediction method in the next time. The proposed approach includes Three Phases: Bugs/Defects classification, System Establishment and setup, Data generation and Dataset building.

2. LITERATURE REVIEW

To inject faults, researchers and engineers have created several new methods that can be implemented in hardware and software. Software fault injection techniques attained by quality control techniques employed during the design and manufacturing of hardware and software. Recently, the awareness that the dependability of distributed systems needs in-depth assessment has pervaded major web enterprises.

By working on injecting a fault for operating systems level, Kao et al (1993) [12] and Broadwell et al. (2002) [13] work on Unix operating system. Where Kao et al presented a new tool (FIME-Fault Injection and Monitoring Environment) that is a first tool used for code changes through the faults of a target program in the executable code. FIME is designed to evaluate of UNIX operating systems based on ODC (Orthogonal Defect Classification) and also designed to assess an open-source operating system (UNIX) in the being of hardware and software errors based on ODC [14]. While Broadwell presented the FIG tool (Fault Injection in Glibc), FIG is a testing tool for UNIX development by C library and desktop robustness/server applications in a UNIX environment. In terms of the impact of faults in an operating system environment, Moreno et al. (2019) [15] present an open source fault injection tool-MiFIT performed on microcontrollers based on timer interruptions.

About injected of distributed system level, researchers in [16] and [17] are work on this level using different methods, where Vedder (2015) [16] applied of Portability based testing method by eveloped the FaultCheck fault injection tool that enables Portability based testing QuickCheck tools to use common fault injection directly on the source code of the complex system. They apply FaultCheck with QuickCheck on both the End to End library and the quadcopter simulator. At (2019), Fibich et al. [17] implement Injected of Distributed System Level done by runtime testing method, where Fibich et al. propose a framework of fault injection based on Netlist faults, Fault InJection Instrumente (FIJI) that can target individual nets at test runtime. Depending on distributed systems, Cotroneo et al. (2019) [18] present the FailViz tool (fault injection visualize) implemented for OpenStack by the anomaly detection algorithm, in their approach distributed systems analysis as a black-box collection interacting during service interfaces.

While some researches done their work depend on the source of code program such as Yu et al. (2020) [19] and Bures et al. (2020) [20], where Yu et al. [19] developed a fault injection tool targeting programs written in C language. This tool relies on exploring sites where faults can be injected by analyzing the source code and then generating failures. But Bures et al. [20] presented defect injection framework(testbed) for benchmark testing, it is a generalized method that has been introduced that does not depend on changing or mutation of the source code, it's based on adding industrial defects to the source code for the target program. The authors here cover a lot of user interface functions by creating test cases (defects) that were used in the test.

Through studying the researches about injecting faults, some of them did not work on one level separately, but on several levels that maybe two or three levels at once. Using faults simulation method, Kaddachi et al. (2016) [21], Amarnath et al. (2018) [22],and Ying et al.(2019) [23] working on operating system and distributed system levels. Where Kaddachi et al. propose a fault injection method provided an injecting of bit flips at target application in data and instructions for realistic faults simulation that occurring in the memory units of the system. By operating on the main source code of the software, a single fault injections simulation does not need hardware. That approach is active with an operating system level and RAM or caches are not set, for that cause, they built an emulator of system memory. While Amarnath et al. presented a framework fault injection (QEMU) based on bit flips simulation of x86 registers during the execution of the system calls of Linux 4.10. At (2019) Ying et al. [23] designed B-SEFI tool to simulate soft error, they applied that tool on five machine learning programs and analyze the programs' weakness to soft errors by simulating bit flips. CentOS operation system is an experimental environment for this tool.

In (2019) again, Porpodas [24] present a new tool for injected the faults based on the timing that is used in fault-coverage studies for transient faults. ZOFI is a zero-overhead tool, meaning that the analyzed workload runs at local speed. While Android devices are an environment operating system for Cotroneo et al. (2019) [25], where he present AndroFIT fault injection tool to apply such fault model on Android devices. This tool is implemented throughout components at the lower layers of the Android OS. 27 components of Android tend to be bug-prone such as OS Services, native components, and device drivers.

According of three levels Injecting (operating system, distributing system, and interface), Netti et al (2018) [26] presented a new Fault injection tool (FINJ)for high performance computing in Python that is implemented as an oriented object, with high level programming language, that is used on many operating systems majors. Among the most significant methodologies utilized fundamentally in defects prediction, algorithms of machine learning classification.

There are many studies use of machine learning techniques for software faults prediction, the related researches examine more machine learning mechanism on several faults datasets, for examples, depending on Mandlebugs dataset, [27] and [28] are predicate fault software by machine learning techniques, where Carrozza et al. used Mandelbugs location in the software of complex systems in their work and fault tolerance mechanisms. They analyze Mandelbugs and discuss a method based on a set of software complexity metrics for Mandelbug prediction. Decision Trees, Bayesian Networks, Support Vector Machines, Naive Bayes, and Multinomial Logistic Regression classifiers are used in that research. For SVM, the predicated correlation coefficient of predicted rating is greater of 70%, and 60% of Mandelbugs can be detected, while regression model determined the rate of 83% of Mandelbugs of exemplary prediction. While Xiaoting et al. focus on the classification of Mandelbugs,and analyzed the impact factors of classification to improve the quality of classification, their method performs well in automatic classifying bugs into fault trigger classes.

In order for different dataset such as CM1 and KC, Surendra and Geethanjali(2013) [4] classifying the faults using decision tree, that work implemented in JAVA by NETBEANS version 7.2. Faults are classified considering 5 attributes Volume, Program length, Difficulty, Effort, and Time Estimator. An Accuracy of their proposed method is 90.77% for CM1 dataset and 90.40% for KC1 dataset. Using NASA MDP dataset, Singh and Shrish (2014) [29] classifying that dataset by cluster-based classification, an evaluate of this method is performed a global relative analysis with benchmark results of software fault prediction for the same data sets. Their proposed model obtained detection probability close to 83.3% and 685% of balance rates.

Another dataset are classified by Ivano(2016) [30] in his thesis, where Ivano work on the failure dataset the faults produce by inject the faults in a several systems contains the OS kernel functions during several runs. New dataset including three attributes of data are collected as Golden Data, Failure Data, and Non-Failure Data, and organized into datasets used for training and validating failure prediction models. During training steps, SVM prediction algorithm is used, for different values in the sliding window. In the experimental, the SVM classifier was execution on the main goal system of ROC-AUC, but on failure mode Hang, the number of the Crash failure events not allow an analysis of the predict response.

When we talking about bug reports classification based on binary classes (Bug, Non-bug), Pingclasai et al. (2013) [31] and Antoniol et al. (2018) [32] works in the same field to classifying of bug reports, where Pingclasai et al. proposed a classification approach to identify bugs and non-bugs. They used Latent Dirichlet allocation (LDA) method with NB and LLR classifier. The precision of HTTPClient, Jackrabbit and Lucene projects varied from 66% to 76%, 65% to 77% and 71% to 82% respectively. But Antoniol et al. work to develop an automatic bug severity approach to detect the bug report whether it is a real bug or request. The features are extracted by Active Directory tree (ADT) and vector space technique. The training is given by Naïve Bayes (NB) and Linear Logistic regression (LLR) algorithms. The precision of Mozilla, Eclipse and JBoss projects interfered with 77% to 82%.

While Kukkar et al. (2019) [33] work on multiclass concept, Kukkar et al. developed a deep learning model for multiclass severity classification called Bug Severity classification, that by using a Convolutional Neural Network and Random forest with Boosting (BCR). This model directly learns the latent and highly representative features. Initially, the natural language techniques preprocess the bug report text, the Convolutional Neural Network extracts the important feature patterns of respective severity classes, and the random forest with boosting classifies the multiple bug severity classes. The average accuracy of the proposed model is 96.34% on the multiclass severity of five open-source projects. The average F-measures of

BCR is 96.43%. Using unstandardized dataset, Hammouri et al.(2018) [34] are used three datasets to predication process by ANN, Naïve Bayes, and Decision Tree supervisor machine learning algorithms, the rate account for the accuracy value in three datasets for the presented algorithms is close to 93%, and the lowest value appears for Naïve Bayes classifier.

3. DEFECTS CLASSIFICATION

An unforeseen software behavior sensed by the users on the software system limits cause to Software failure is a faulty consequence, while a software fault is the specific or supposition cause of that software failure.When the uniqueness fault and failure is not stringent that can be named as a defect to mention to either a cause or an effect. Chillarege and co-workers [35] provided ODC, a complete and practical classification of software defects from various perspectives.

Experiences have shown that every software systems hold faults such as bugs or defects, except easy software systems. [36]. For characterizing of software faults that cause failures through experiment and execution, developers and researchers pint to “Bohrbugs,” “Heisenbugs,” “Mandelbugs,” and “aging-related bugs.”. Grottke et al. suggest bugs definitions of the situations and state regarding the fault activity and the error diffusions [36, 37].

4. PROPOSED METHODOLOGY

Because the process of injecting a real error or fault is very expensive for computer parts of software and hardware, we have relied on learning of open-source software and their preparations to receive faults and predict them through these faults reports in the event that one of these errors/faults target the systems really during the system execution period. Through the process of research and study of tools previously built to inject faults that were published in our review of related works [8], there was a need to build a new injection tool through which we can inject faults to an integrated system that includes an operating system, a web browser, database software, and the web language software.

Fault Injection for Bug Reports in Open-source software (FIBR-OSS) is based on a library of bug reports for open source software, the defects of complex software that are contained in the FIBR-OSS library are dependent on a global field study to identify the bugs list that can rationally be predictable to occur much of failures in OSS's. Our proposed system includes Three Phases: Bugs/Defects classification, System Establishment and setup, Data generation and Dataset building.

4.1. First phase: bugs/defects classification

In order to train the four open-source software which is integrated in the system that will be injected with faults in order to assess the dependability of this system, we have combined 12 datasets for those open-source software in single dataset (General dataset) to be used in the process of system environment training, the purpose of which is to train that environment contains the fault type and the fault location in any four open-source software mentioned earlier. In this context, and based on classification algorithms in order to predict faults or defects in the open-source software, we present a software defect prediction model by LMT machine learning and DNN on new dataset combined from 12 public datasets for software open-source Linux kernel, MySQL DBMS, Apache HTTPD web server, and Apache AXIS WS available in [38] that dataset is a general dataset. The bugs have existed classified based on faults that occur on open-source software and attributed to Bohrbug, Mandelbu, Aging Related Bug, and Unknown bugs [39]. Both Deep Learning and Logistic Model Tree classifiers are evaluated based on the primary source of performance measurements, which is a confusion matrix and a set of most-known measures [40, 41].

4.2. Second phase: system establishment and setup

In this phase, one must define the requirements, network environment to build our prototype of software fault injection model, and Target System Training. As shown in Figure 1, virtual machine setup and configuration for two systems: the first is the Clint, which is the system that receives the faults (i.e. the target systems), whose dependability will be measured, the second is the Server, which is the system that injects the faults.

We are building a new model for fault injection in open source software, and the requirements for this model are Network environment, Virtual machine or online network.

Components of the system are:

a) Target Systems (Client), in our work we suggest four open-source software
b) Source System (Server)
c) Bug's Library (bugs and defects) dataset.

Databases (library) of errors affecting open source software (study case) which are Linux kernel, MySQL DBMS, Apache HTTPD web server, and Apache AXIS WS.

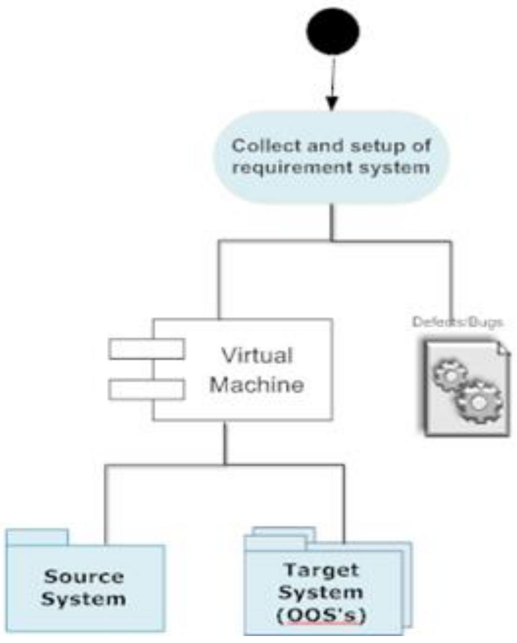


Figure 1. System establishment and setup

4.3. Third phase: data generation and dataset building

Software faults injection is a robust machine to support and improves failure prediction techniques. This is an interest research matter in the assessment dependability of systems because of the complexity increasing of open-source software, which makes software failures most happened.

A new prototype of the fault injection model is presented, FIBR-OSS (Fault Injection for Bug Reports in Open-Source Software). FIBR-OSS Faults injection model is shown in Figure 2, the model contains injector software, injection controller software, and monitoring procedures to collect data to use it at the dependability assessment of the target system. Classification accuracy for the classifiers DNN and LMT as shown in Table 1.

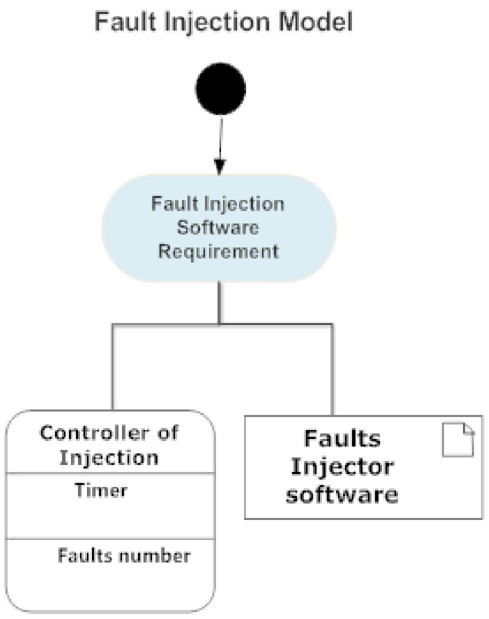


Figure 2. FIBR-OSS: Faults injection model

Table 1. Classification accuracy for the classifiers DNN and LMT

Performance Measures	DNN Classifier Classes					LMT Classifier Classes				
	BOH	NAM	ARB	UNK	Weighted Av.	BOH	NAM	ARB	UNK	Weighted Av.
TPR	0.995	0.752	0.988	0.0	0.917	0.993	0.695	0.983	0.0	0.877
Precision	0.887	0.983	1	0.0	0.873	0.846	0.961	1	0.0	0.849
Recall	0.995	0.752	0.988	0.0	0.917	0.993	0.695	0.983	0.0	0.877
F-measure	0.962	0.922	0.986	0.0	0.911	0.913	0.807	0.991	0.0	0.854

1) Injector: A new software injector is developed using the defects datasets that collected as one of the requirements in the previous phase.
2) The controller of fault injection is software integrated with an injector to determine slot time injection and fault size (Number of Bugs).

To collect data through the different time periods, the monitoring procedures collect the values of the variables of the system state. Then that collected data are organized in new datasets used later under the failure prediction model.

A new data are collected through various time intervals, the monitoring software collects the values of the variables describe the target system state, this is a core phase of our approach, where the data are collected while the target system executes and faults are injected by our new fault injection model.

FIBR-OOS model works depending on the system's training of the bug's reports before Bugs injection operation, Bug Report Injection and New Dataset Generation and Building Algorithms are summarizing as follow.

Bug Reports Injection and New Dataset Generation Algorithm:

1. Read Bug Report Dataset
2. Determine N Value, *t* Value, where N is Number of Bug data that will be injected, T is Time period for injection (*t* sec., *t* min., etc.)
3. Randomly by Injector software, select N Bug data from General Dataset (Library)
4. For T = 0 into (*t*-1) do:
   Pass N bug data from (Server machine) to (Clint machine)
5. During Step 2 running, Update General Dataset for any ID-Bug:
   a. New Attribute Column as System State,
      If System is crashing then System State = Failure
      If System is Hanging then System State = Critical
      If System is still Running then System State = Normal
      Else System State = Unknown
   b. New Attribute Column as System Response,
      System Observation: Depending on System State and Part of Open-source software that will be injected.
   c. New Attribute Column as Expected Solution,

      Expected solutions are depending on the system state.

The studying and monitoring of system's behavior during faultload time, creating a new dataset include of system state:

1. Failure corresponds to fund complex bug that making system Crashing/Failure.
2. Critical corresponds to fund less complex bug making system hanging.
3. Normal corresponds to inject a simple bug that hasn't effect on system execution/running.
4. Unknown state when an unknown bug is injected in the system.

Figure 3 shows the Fault injection environment that used for new data generation and dataset collection on Open-Source Software System. Accuracy Measures for Deep learning and LMT classifiers as shown in Figure 4.

What happens during the transmission is:

1. The data at the time of transmission (injection) will be a data set of three columns (attributes) ID, Bug name, and MandleBug Classes.
2. The Server Machine needs an IP + port addresses.
3. The Client Machine only needs the port address.
4. The data sent is text, so if the data is sent as a text every time, there will be a load status on the transmission channel and thus will have some data loss.
5. Solution: convert the text into File (Packets) and then treat it as bytes

During the preset injection times in the error injection algorithm, new data are collected depending on the system status during the injection. The General Dataset was updated during a number of injection operations and reached about 1000 times, which took place during different time periods. The injection, monitoring, and data collection process took a period of up to 3 months, where the injection was made for the following time periods:

1. 10 seconds and 10 Bug randomly chosen
2. 50 seconds and 100 Bug randomly selected
3. 5 minutes, 300 Bug chosen randomly

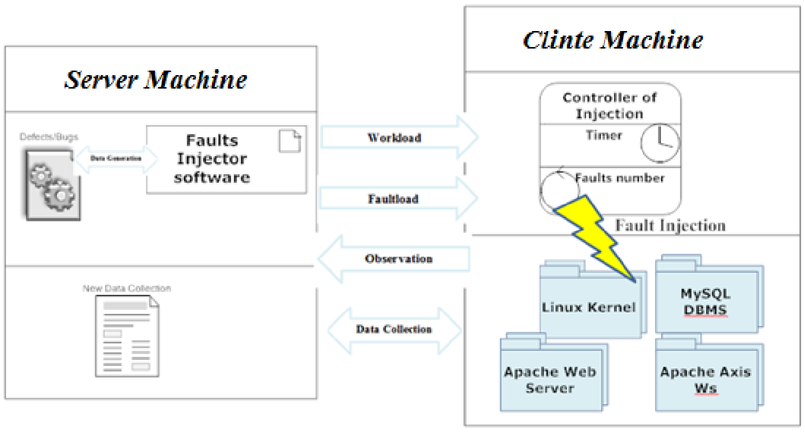


Figure 3. Fault injection environment and dataset collection

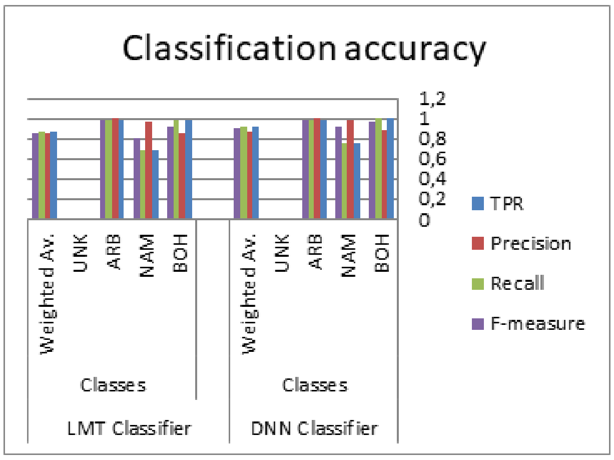
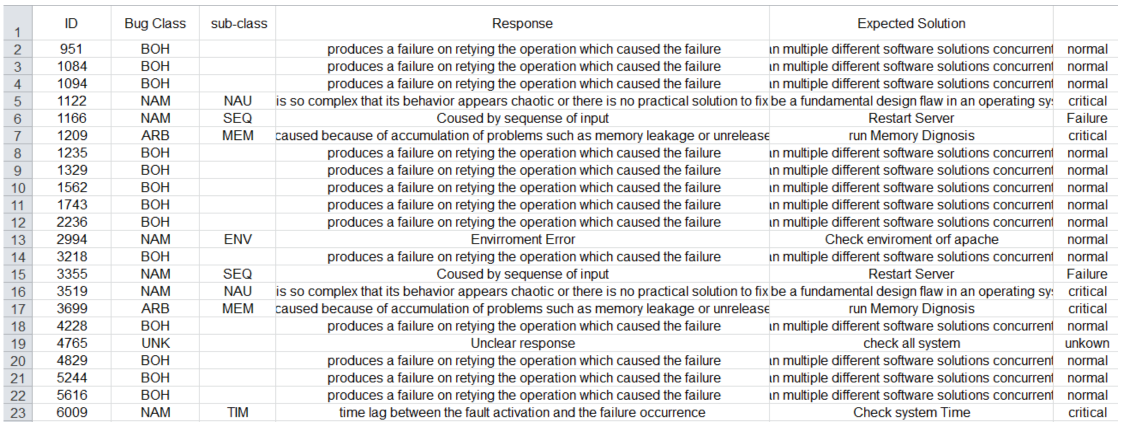


Figure 4. Accuracy measures for deep learning and LMT classifiers

Note that the user can control the injector in terms of time and number of Bugs sent (injected) into client machine (system target). For the purpose of studying the proposed system states, we will focus here on two types of open-source software Linux kernel and Apache HTTPD web server because it is the most provided in the use. Table 2 shows a part of the data collection collected during the injection process and the generation of new dataset for Linux kernel. Where BOH = Bohrbugs, MAN = Mandelbugs, ARB = aging-related bugs, and UNK = Unknown bug.

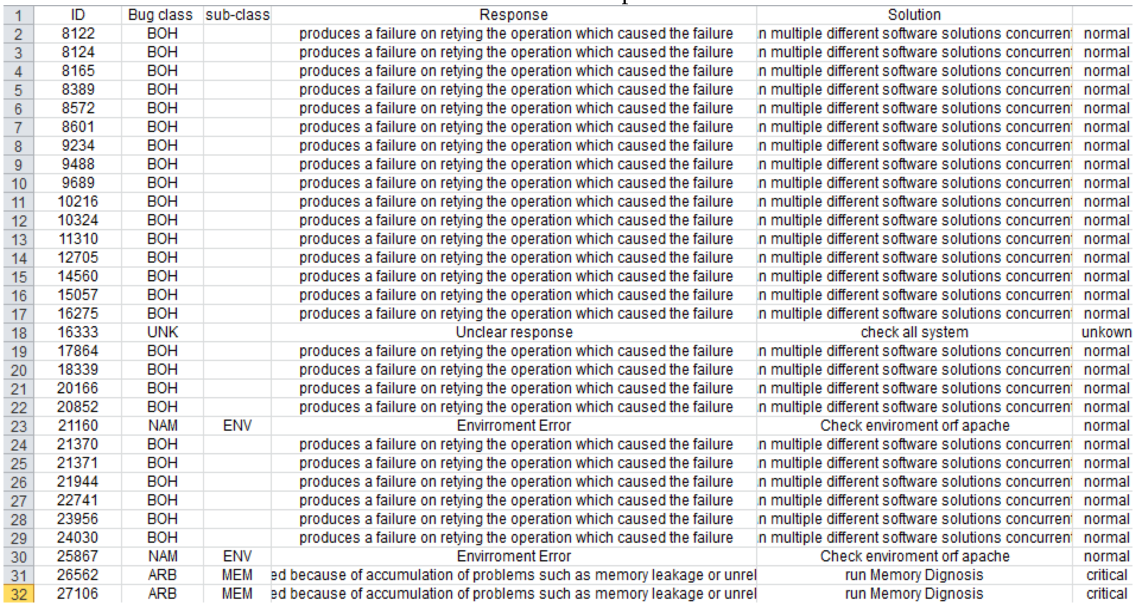
As shown in Table 2, Linux kernel has a little failures state in the case if the Bug-type is NAM caused by wrong inputs continuously, while the critical state of the system is more if compared to cases of a failure state, which often causes by loading memory more than the real absorption. For her, it was noted that most states are normal when the Bug-type is BOH and this means exactly the principle of fault tolerance as the system continues to run despite the presence of faults until a fault occurs in which the system should be stopped. Table 3 shows a part of the data collection collected during the injection process and the generation of new dataset for Apache HTTPD web server.

Table 2. New dataset collected for linux kernel software



1	ID	Bug Class	sub-class	Response	Expected Solution	
2	951	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
3	1084	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
4	1094	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
5	1122	NAM	NAU	is so complex that its behavior appears chaotic or there is no practical solution to fix	be a fundamental design flaw in an operating sy:	critical
6	1166	NAM	SEQ	Coused by sequense of input	Restart Server	Failure
7	1209	ARB	MEM	caused because of accumulation of problems such as memory leakage or unrelease	run Memory Dignosis	critical
8	1235	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
9	1329	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
10	1562	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
11	1743	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
12	2236	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
13	2994	NAM	ENV	Envirroment Error	Check enviroment orf apache	normal
14	3218	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
15	3355	NAM	SEQ	Coused by sequense of input	Restart Server	Failure
16	3519	NAM	NAU	is so complex that its behavior appears chaotic or there is no practical solution to fix	be a fundamental design flaw in an operating sy:	critical
17	3699	ARB	MEM	caused because of accumulation of problems such as memory leakage or unrelease	run Memory Dignosis	critical
18	4228	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
19	4765	UNK		Unclear response	check all system	unkown
20	4829	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
21	5244	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
22	5616	BOH		produces a failure on retying the operation which caused the failure	ın multiple different software solutions concurrent	normal
23	6009	NAM	TIM	time lag between the fault activation and the failure occurrence	Check system Time	critical

Table 3. New dataset collected for apache HTTPD web server



1	ID	Bug class	sub-class	Response	Solution	
2	8122	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
3	8124	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
4	8165	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
5	8389	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
6	8572	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
7	8601	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
8	9234	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
9	9488	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
10	9689	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
11	10216	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
12	10324	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
13	11310	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
14	12705	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
15	14560	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
16	15057	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
17	16275	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
18	16333	UNK		Unclear response	check all system	unkown
19	17864	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
20	18339	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
21	20166	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
22	20852	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
23	21160	NAM	ENV	Envirroment Error	Check enviroment orf apache	normal
24	21370	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
25	21371	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
26	21944	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
27	22741	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
28	23956	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
29	24030	BOH		produces a failure on retying the operation which caused the failure	n multiple different software solutions concurren	normal
30	25867	NAM	ENV	Envirroment Error	Check enviroment orf apache	normal
31	26562	ARB	MEM	ed because of accumulation of problems such as memory leakage or unrel	run Memory Dignosis	critical
32	27106	ARB	MEM	ed because of accumulation of problems such as memory leakage or unrel	run Memory Dignosis	critical

As shown in Table 3, Apache HTTPD web server has a little critical state of the system is more if compared to cases of a another software in our system, which often causes by loading memory more than the real absorption. While Failure state is almost non-existent and rare, and this means exactly the principle of fault tolerance as the system continues to run despite the presence of faults until a fault occurs in which the system should be stopped.

5. CONCLUSION

Any failure prediction models depend on behavior observations of the target system during past times, which can be express by numeral time-series data or class and subclass data as store events in a log file. In this research, we propose the use of existing features obtained by bug tracking systems, without any specific one of these features. And also, we motivate to using numeral data instead of log files data, as it has been demonstrated that categorical data may degrade the performance of a prediction model.

The choice of data type used in this work depended on the mechanism of data collection during the injection stages of faults, and this is what urged us to use the log files data type although difficult to deal with. In order to improve more accuracy, a new general dataset has been trained using Deep Neural Network and LMT again on a new combined dataset as general dataset. Fault and defect predictors using the DNN classifier performs better than LMT where LMT accuracy near from 0.849, while DNN accuracy is 0.873. In terms of speed and better results, the major cause to make deep learning the best choice is that it has given better performance on many complex problems.

FIBR-OSS can support developers to evaluate the system performance such as reliability and dependability means such as fault prediction or forecasting during phases development. FIBR-OSS is used for fault speed-up to test the system's failure prediction performance. During the preset injection times in the error injection algorithm, new data were collected depending on the system status during the injection. The new set of data will then be used in the failure prediction process for open-source software. It is worth noting that we recommend the use of a Deep learning algorithm in the failure prediction process, which saves a lot of costs if we try another algorithm as it has proven the accuracy of its results

ACKNOWLEDGMENT

The author's thankful Department of Computer Science, Collage of Science, Mustansiriyah University, University of Technology, and Informatics Institute for Postgraduate Studies, for supporting this work.

REFERENCES

[1] N. Ullah, M. Morisio, and A. Vetro, "A comparative analysis of software reliability growth models using defects data of closed and open source software," *2012 35th Annual IEEE Software Engineering Workshop*, pp. 187-192, 2012.
[2] R. Natella, "Achieving Representative Faultloads in Software Fault Injection," University of Naples Federico II, Italy, 2011.
[3] N. Kalaivani and R. Beena, "Overview of software defect prediction using machine learning algorithms," *International Journal of Pure and Applied Mathematics*, vol. 118, pp. 3863-3873, 2018.
[4] M. S. Naidu and N. Geethanjali, "Classification of defects in software using decision tree algorithm," *International Journal of Engineering Science and Technology,* vol. 5, p. 1332, 2013.
[5] Y. Tamura and S. Yamada, "Reliability and maintainability analysis and its toolbased on deep learning for fault big data," in *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 106-111, 2017.
[6] L. Feinbube, L. Pirl, and A. Polze, "Software Fault Injection: A Practical Perspective," in *Dependability Engineering*, ed: IntechOpen, 2017.
[7] A. Okutan, "Software defect prediction using bayesian networks and kernel methods," ISIK UNIVERSITY, 2012.
[8] S. A. Alazawi and M. N. Al-Salam, "Review of Dependability Assessment of Computing System with Software Fault-Injection Tools," *Journal of Southwest Jiaotong University,* vol. 54, 2019.
[9] R. Natella, S. Winter, D. Cotroneo, and N. Suri, "Analyzing the Effects of Bugs on Software Interfaces," *IEEE Transactions on Software Engineering,* 2018.
[10] K. Umadevi and S. Rajakumari, "A review on software fault injection methods and tools," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 3, pp. 1582-1587, 2015.
[11] A. Avižienis, J.-C. Laprie, and B. Randell, "Dependability and its threats: a taxonomy," in *Building the Information Society,* ed: Springer, pp. 91-120, 2004.
[12] W.-I. Kao, R. K. Iyer, and D. J. I. T. o. S. E. Tang, "FINE: A fault injection and monitoring environment for tracing the UNIX system behavior under faults," vol. 19, pp. 1105-1118, 1993.
[13] P. Broadwell, N. Sastry, and J. Traupman, "FIG: A prototype tool for online verification of recovery mechanisms," in *Workshop on Self-Healing, Adaptive and Self-Managed Systems*, 2002.
[14] M. M. Manhães, M. C. P. Emer, and L. C. Bastos, "Classifying defects in software maintenance to support decisions using hierarchical ODC," *Anais do Computer on the Beach,* pp. 283-292, 2014.
[15] A. Aponte-Moreno, F. Restrepo-Calle, and C. Pedraza, "MiFIT: A Fault Injection Tool to Validate the Reliability of Microprocessors," *2019 IEEE Latin American Test Symposium (LATS),* pp. 1-5, 2019.
[16] B. Vedder, "Testing Safety-Critical Systems using Fault Injection and Property-Based Testing," Halmstad University Press, 2015.
[17] C. Fibich, S. Tauner, P. Rössler, M. Horauer, M. Matschnig, and H. Taucher, "FIJI: Fault InJection Instrumenter," *EURASIP Journal on Embedded Systems*, vol. 2019, p. 2, 2019.
[18] D. Cotroneo, L. De Simone, P. Liguori, R. Natella, and N. Bidokhti, "FailViz: A Tool for Visualizing Fault Injection Experiments in Distributed Systems," *2019 15th European Dependable Computing Conference (EDCC),* pp. 145-148, 2019.
[19] H. Yu, H. Gong, and Y. Wang, "Design and implementation of fault injection based on abstract syntax tree of C Program," presented at the IOP Conference Series: Materials Science and Engineering, 2020.
[20] M. Bures, P. Herout, and B. S. Ahmed, "Open-source Defect Injection Benchmark Testbed for the Evaluation of Testing," *arXiv preprint arXiv:2001.09342,* 2020.
[21] F. Kaddachi, M. Kooli, G. Di Natale, A. Bosio, M. Ebrahimi, and M. Tahoori, "System-level reliability evaluation through cache-aware software-based fault injection," *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS),* pp. 1-6, 2016.
[22] R. Amarnath, S. N. Bhat, P. Munk, and E. Thaden, "A fault injection approach to evaluate soft-error dependability of system calls," *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* pp. 71-76, 2018.

[23] Y. Wang, J. Dong, S. Zhang, and D. Zuo, "B-SEFI: A Binary Level Soft Error Fault Injection Tool," *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C),* pp. 235-241, 2019.
[24] V. Porpodas, "ZOFI: Zero-Overhead Fault Injection Tool for Fast Transient Fault Coverage Analysis," *arXiv preprint arXiv:1906.09390,* 2019.
[25] D. Cotroneo, A. K. Iannillo, R. Natella, and S. Rosiello, "Dependability Assessment of the Android OS Through Fault Injection," *IEEE Transactions on Reliability,* 2019.
[26] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, and A. Borghesi, "FINJ: A fault injection tool for HPC systems," *European Conference on Parallel Processing,* pp. 800-812, 2018.
[27] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Analysis and prediction of mandelbugs in an industrial software system," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 262-271, 2013.
[28] X. Du, Z. Zheng, G. Xiao, and B. Yin, "The automatic classification of fault trigger based bug report," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 259-265, 2017.
[29] P. Singh and S. Verma, "An efficient software fault prediction model using cluster based classification," *Int. J. Appl. Inf. Syst,* vol. 7, pp. 35-41, 2014.
[30] I. Irrera, "Fault Injection for Online Failure Prediction Assessment and Improvement," 2016.
[31] N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *2013 20Th asia-pacific software engineering conference (APSEC),* pp. 13-18, 2013.
[32] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement? A text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pp. 304-318, 2008.
[33] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A Novel Deep-Learning-Based Bug Severity Classification Technique Using Convolutional Neural Networks and Random Forest with Boosting," *Sensors*, vol. 19, p. 2964, 2019.
[34] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *IJACSA) International Journal of Advanced Computer Science and Applications,* vol. 9, 2018.
[35] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, *et al.*, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on software Engineering,* pp. 943-956, 1992.
[36] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," *2010 IEEE/IFIP international conference on dependable systems & networks (DSN),* pp. 447-456, 2010.
[37] M. L. Shooman, "Software reliability growth model based on Bohrbugs and Mandelbugs," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 381-386, 2015.
[38] R. Natella. Mandelbugs in Open-Source Software [Online].
[39] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Analysis and prediction of mandelbugs in an industrial software system," *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation,* pp. 262-271, 2013.
[40] D. L. Olson and D. Delen, *Advanced data mining techniques*: Springer Science & Business Media, 2008.
[41] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann, 2016.