REST Journal on Emerging Trends in Modelling and Manufacturing
Vol: 9(3), September 2023
REST Publisher; ISSN: 2455-4537
Website: http://restpublisher.com/journals/jemm/
DOI: https://doi.org/10.46632/ jemm /9/3/6

Debugging the Localization Bugs and Improve Microprocessor Performance Using Machine Learning

***R. Samson Ravindran, P. Balashanmuga Vadivu, N. Jayanthi, Suresh Kumar Balam**
Mahendra Engineering College, Namakkal, Tamil Nadu, India.
*Corresponding author Email: ed@mahendra.org

Abstract: *Processor design evaluation and debugging are challenging and complex tasks that take up the majority of the design process as well as require significant engineering resources. The performance of overall processor gets reduced because of bugs with no effect of the functionality are especially challenging for debugging owing to the lack of a universal standard from bug-free performance. This is due to the functional errors, the optimum processor performance for novel microarchitectures on complicated whereas the benchmark of long-run is usually predictably determined. To solve the mentioned problems, we are presenting a Machine Learning (ML) approach that is Deep Neural Network (DNN) with Improved Firefly Algorithm (IFA) as DNN-IFA as a classifier for performance benchmarking of new microarchitectures. The performance of a novel microarchitecture may be assumed to be appropriate if the present microarchitecture outperforms the preceding generation, despite considerable performance regressions in the initial implementation. Moreover, this proposal has influences hyperparamer and by modifying the feature fitness of firefly algorithm resulted to improve the fitness by avoiding bias has capability in detecting the bug performance endurance in microprocessors. The findings reveal that the most effective technique discovers microprocessor core performance defect is 91.5% with a standard IPC impact of more than 1% across the analyzed applications, compared to a bug-free design with no instances of false positives. The suggested system in the simulated scenario takes consumes less time to execute a bug location inference that resulted to minimize the debugging time.*

Keywords: *Microprocessor, extra tree classifier, microarchitecture, debugging, localization bug, Machine Learning*

1. INTRODUCTION

Microprocessors development is an immediate consequence of Bell Labs' semiconductor transistors conception and Robert Noyce's design of Integrated Circuit (IC) chips. During 1969, the calculator manufacturer named Busicom has approached Intel for manufacturing 12 chips for calculators. In response, Intel created four designs, these had programmed in various ways for meeting the client needs. Intel 4004 is the design that had generated as well as released its chipset at 1971 which was the first microprocessor despite its limited capabilities included only primitive 4-bit arithmetic functions. In the meantime, Texas Instruments (TI) submitted a patent for microprocessors, and issued at 1973. In 1974, TI unveiled TMS1000 as the first microprocessor and the specification is 4-bit with 1 KB ROM and 32 byte RAM. This triggered an avalanche of advancements and research in this particular field. Continued effort in improving IC technology resulted in a varied spectrum of 8-bit and 16-bit microprocessors over the subsequent few decades. The Intel 4004 was succeeded by the subsequent Intel 8008 as well as 8080 microprocessors. It was subsequently upgraded to Intel 8085, which had more guidelines, interrupts, including serial input/output (I/O) pins. The emergence of personal computers paved the way for a new era of microprocessors. Furthermore, emerging complementary metal oxide semiconductor (CMOS) technology supported Moore's Law and played an important role in shaping new microprocessor architectures. Mead and Conway's research in very large-scale integration (VLSI) pioneered novel design techniques in both academia and business. Several computer-aided design (CAD) as well as simulation programs has been created, including those from

Cadence, Synopsys, Mentor Graphics, and others, for creating designs and analyzing VLSI circuits for different performance levels [1]. This allowed designers to investigate novel design techniques and assess their performance throughout the designing stage, resulting in advancements to microprocessor architecture.

Each microprocessor design endeavor requires a significant time consumption and work for verification as well as validation. Generally, design verification is often divided into two major groups are

1. Functional
2. Performance verification

Its purpose is to detect design flaws that reduce performance without harming functionality. Performance bugs differ from performance bottlenecks in that the former is caused by design errors, whilst the latter is caused by limited resources. Performance-related problems can cause significant performance loss with recent reports indicating that it is greater than 10% [2]. Performance evaluation at the microarchitecture level verifies that a design meets the desired performance with regard to execution period or cycle count. The primary problem concerning the study is that, unlike functional verification, there is no exact gold standard to compare against. This is due to the difficulties of adequately representing how all of the interactions between the various parts in complicated microprocessor designs affect the overall system performance. Traditionally, performance verification has been done primarily using manual procedures that rely on rough estimates of the performance gains expected from microarchitectural improvements. Although advances in testing design techniques and processes, we continue to notice a large number of post-production problems once new microprocessors are deployed, showing that there are gaps in design testing and validation [3, 4]. Before being sent to clients, a microprocessor undergoes a series of evaluation and certification stages. In the initial phases, a design simulation with random or human-generated inputs could identify flaws. Once the design has matured, informal methods for verification are used to guarantee that selected design parts are valid. In this work, identifying these gaps and recommends specific methods to close gaps using an entirely novel classification with respect to defects given by Intel and AMD.

Bug localization has become the most critical duties in software development. Defect localization is a method of identifying a defect or error in software which occurred during operation. Manual bug locating is costlier and high time consumption in operational [5]. Several automated bug localization approaches are used to detect problems in software. Bugs are localized using various software artefacts including source code, previous bug reports, and test cases. Bug reports is a report describing the issues found in the software that includes a bug summary, definition, stack traces, as well as bug metadata, including information about whoever reported the bug and how they fixed it.
Some solutions use bug report text directly to locate the bug using information retrieval techniques [6], [7]. Such strategies used similarity measurements between bug report text and various source code files. The obtained similarity scores for source code files are ordered in descending order to discover the source code file that is most similar to the issue report. There have been few previous studies on automated detection of microprocessor performance issues. The majority of these relies on preserving design objectives employing a customized performance model as a perfect reference, which requires a significant development period as well as include errors on their own. Data-driven and ML technique to automate bug detection performance with excellent accuracy has recently been developed [8]. Although relevant, these studies do not address the bug localization performance issues. However, the bug localization performance is a difficult task that is being performed mostly in manual. Therefore, several studies illustrate the functional validation in which the organization was lacking in standard guidelines for automating the bug localization. Thus, the research paper focus on generating the task in automates the bug localization. Moreover, the proposed DNN with firely fitness algorithm has concentrated in designing the automation that fortunately minimize the time consumption as well as performance needed by engineers for debugging its design with highly valuable.

2. LITERATURE REVIEW

Considering a preliminary effort on bug localization performance, the study design has possible limits and nonetheless believes that it offers a good beginning step in tackling the issue. To investigate the research questions, it use the technique with the JDT as well as Eclipse datasets described. All dataset utilize 90% as training data as well as 10% as testing data. As a result, certain of the previously corrected bugs shall be utilized to determine whether the method utilizing correctly recommends the necessary source code files that require to be fixed. To determine the correctness of the method across any study issues, each dataset will be divided by tenfold cross-validation in several previous experiments [9] [10]. The ten folds have been separated into nine to train, with one left

unfixed to test. This technique can be repeated ten times, with one of the ten folds utilized to test and the rest to train. The outcomes of all folds are then averaged for Top-1, 5, 10, and 20 measurements. A system proposed [11] that use a deep learning algorithm for bug localization. Bug reports with clauses are used beside source code file characteristics. The contributions are accurately assessed on four datasets are AspectJ, Tomcat, SWT, and JDT. The topic analysis method was employed to identify bugs in JINGO [12] by analyzing bug reports as well as source code. Fang et al. [13] aimed to improve the performance of the data extraction technique. This method focused the bug quality report as text, whether relevant or not. An additional investigation looked at the combination of different IR techniques [14] such as VSM and LSI, and the researchers determined that the combination had improved results. An IR approach known as BoostNShift [15] helps to localize the problem at the method level. The authors attempted to use bug report wording as a query with the source code to find techniques that contained problems. Simulation is a typical testing technique which is utilized during the beginning of the design development process. F.Solt et al. illustrated methods to create a simulation using input sequences. The outputs and the final state of the design are subsequently compared to a conceptual model or inspected manually [16]. J.Balkind et al. discusses the current simulators include a wide range of capabilities, including undefined values are signal injection, and numerous coverage measures. There are two significant drawbacks of simulation testing. Firstly, simulation-based testing is exceedingly slow. As a result, it is capable of processing certain inputs in an acceptable amount of time, making it difficult to achieve all potential system states. The open-source CVA-6 64-bit RISC-V CPU takes four days to load Linux in simulation is an instance [17]. More complicated CPUs designed for high performance are expected to be several orders of magnitude more complex. Hector, developed by Saha et al., searches for resource release omission faults in error-handling code statically. This is a special type of problem in which an error-handling pipeline is missing several resource release operations. Missed unlock calls, memory frees, etc. Dinghao Liu discusses the main concepts about fundamental blocks near other blocks with resource-release activities may likewise require those operations. IPPO detects defects in security critical paths, especially error paths, employing differential inspection of identical routes [18]. V.Murali et al. have illustrated proposed program as Bug2Commit that uses rapid text as a word embedding to retrieve source code files, metadata, and stack traces in addition to bug reports. The significant distinction with this strategy is that it achieved good accuracy for a dataset that was not a benchmark, but the nature of the software project differed from the Facebook application. Furthermore, the skip gram word embedding model is used to determine the semantic similarity of bug reports and source code documents [19]. Deep neural networks use these qualities to identify the source code files containing the issue. A.Thakahashi et al. have examined their work on five well-known benchmark datasets, including SWT, AspectJ, Eclipse, and JDT, and achieved positive results with an approach known as smellware-based [20]. Code smells can be caused by a design flaw or a fault with the source code. The code smells are used here to improve the process of IR-based. S.Sangle illustrates the proposed approach named DRAST tries to improve the flexibility of bug localization so that it can handle several languages. In addition to a vector space model, there are random forest and deep learning algorithms. This strategy achieves a 90% increase in MRR and MAP [21].

3. RESEARCH METHODOLOGY

In this research, automated improved ML method is proposed for identifying the bug present in the microarchitectural unit. The term microarchitectural unit represents the microprocessor segment which involves certain task. The proposed method has strengthen the performance counter data as an inputs but available in all kind of microprocessor design. It may prevent the overhead by accumulating assigned infrastructure of data acquisition. Moreover, the design of microarchitecture basically consists of hundreds and thousands of performance counters that are quite adequate for extracting required data to estimate the microprocessor performance. Hence, the DNN-IFA is considered for detection of bug localization.

1. DNN-IFA model for bug localization detection

The proposed DNN-IFA model is utilized for detecting the bug existence is shown in figure 1.

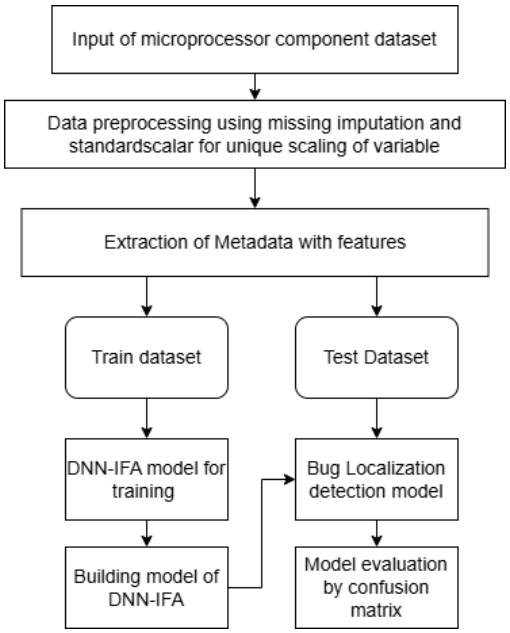

Figure 1. DNN-IFA model for Bug localization detection

The Performance Prediction error-Based Classification (P2BC) approach consists of two distinct ML phases. The first level is a set of performance prediction models built solely on design data with bug-free for capturing the relationship among counters and performance in normal settings. When these models are applied to problematic designs, large prediction errors occur because the optimal conditions no longer exist. The subsequent step utilizes the prediction errors is one of the features for DNN-IFA classifiers. The concept of this method to analyze the particular workloads wherein differ with the model as well as ground truth, alongside the feature divergence, might reveal valuable data regarding which functional unit is causing the bug. As a result, P2BC identifies issues based on evidence of model failure performance. The initial stage of the P2BC approach has been heavily influenced by earlier studies about performance of bug detection but the subsequent stage varies significantly in that it employs the complete error trace for feeding the ML models in performing bug localization, whereas previous work utilized error metric only as input for heuristic rule-based classifier solely to determine bug existence.

3.1 Improved Firefly Algorithm

Fireflies are little bugs providing light at night and its rough wings emit light from luminous compounds in their stomachs. The Firefly Algorithm (FA) is motivated by the flashing characteristic of fireflies. The least brilliant fireflies are drawn to stronger fireflies keeping consideration to the media encircling an issue domain. Getting in a single fewer bright fireflies towards brighter fireflies represents a single iteration. The ideal address has been sought through a step-by-step iteration procedure. The method is to represent each firefly as a vector point. The firefly position is mentioned through candidate solution C_a where a = 1, 2,....9. Firefly attraction and brightness is formulated in equations 1 and 2.

$$f_b = f_{bo} e^{P\forall} ab \tag{1}$$

$$MD_a(\forall) = MD_{ao} e^{\forall} ab^2 \tag{2}$$

Where,

f_b = fluorescence firefly brightness over $\forall$ with assumed as zero.

P = Light absorption parameter.

ab = Firefly position.

MD = Maximum degree of attraction.

Distance among two fireflies has been computed by Euclidean distance is represented in equation 3.

$$\forall_{ab} = \|C_a - C_b\| = \sqrt{\sum_{k=1}^{m}(C_{ak} - C_{bk})^2} \quad (3)$$

Once less bright fireflies begin their trek to stronger fireflies as well as operating speed may have enhanced using substituting equation 4 in equation 2.

$$MD(\forall) = \frac{MD_o}{1 + (P * \forall_{ab^2})} \quad (4)$$

Firefly affecting from position *a* to *b* is measured by formulae shown in equation 5.

$$C_a^{n+1} = C_a^n + \frac{MD_o}{1 + P\forall_{ab}^2}(C_a - C_b) + \varphi(j - .5) \quad (5)$$

Where,
j= random value between 0 and 1.
n = Number of iteration
φ = Step size

Therefore, it analyzed how fireflies move from one spot to an additional as well as how the brightest firefly can be recognized. Typically, the position of the firefly is updated depending on the fluorescence brightness attraction, and the search is performed using random and global parameters. For improving accuracy and find the best answer has suggested an approach that changes the step size on a regular basis. Changing the firefly position on a regular basis reduces the number of search iterations for fluorescence brightness. Equation (6) illustrate about calculating the distance among individual as well as group center to find the diversity of initial group.

$$d^i = \frac{1}{|C_p|}\sum_{a=1}^{|C_p|}\sqrt{\sum_{b=1}^{m}(C_{ab} - \overline{C_b})^2} \quad (6)$$

where,
d^i = Diversity index.
$|s_p|$ = Firefly population size

Number of iterations has improved in decreasing the linear decrease function ω. Equation (7) has employed for calculating the ω with two factors namely maximum iteration number (max_i) and current iteration number $(curr_i)$
.

$$\omega = \frac{max_i - curr_i}{max_i} \quad (7)$$

Working of DNN method

DNNs can learn higher-level characteristics with greater complexity and abstraction. In these applications, values in the features are the input towards DNN initial layer as well as the output layers are comprehend as indicating the existence of various low-level features in the data with records. At subsequent layers, these features are merged into an estimation of the likelihood of higher-level features, such as weight of features are subsequently merged into sets of feature weights. At last, provided all information, the network calculates the probability with high-level traits are present in a specific data.
However, DNN method hierarchy has been enabled with deep feature for outperforming in a variety of tasks. When employing a DNN for inference and provide both input image as well as DNN returns a scores as vector to each object class. Hence, the highest score class is considered to be coherent in classifying the target classes. The primary purpose of DNN training is to find the weights that maximize correct class scores while minimizing incorrect class scores. During network training, the proper classification is frequently known because it is provided for the feature data used for training. The loss (L) is the difference among the ideal accurate scores whereas the scores have been

measured by DNN with respect to weights. Thus, the training DNNs objective is in discovering the weights set that minimize the average loss across a enormous training dataset. When network training, the weights (w_{ij}) are often changed by a hill-climbing optimization algorithm is said to be gradient descent.

The loss related to the multiple gradient of each weight has considered the loss with partial derivatives in term of weights is utilized for weight update.

$$W_{ij}^{t+1} = W_{ij}^{t} - \alpha \frac{\delta L}{\delta W_{ij}} \quad (8)$$

Where,

α = Learning Rate

The purpose of the gradient will represent the change in weight for reducing the loss whereas the process may repeat iteratively for minimizing the overall loss. The gradient partial derivatives has been calculated in an effective manner is said to be backpropagation and it is calculated by derivatives from calculus chain rule that can be functioned through passing the values towards backward of network in computing the loss obtained from each weight.

Bug Localization with error-based

Although the error traces from starting stage collected from multiple workloads with varying execution periods, the time-steps number on each trace may range among workloads. As a result, once obtaining error traces for all workloads, it undergoes a resampling step. The purpose of this process is to modify all of the traces to ensure each trace has a consistent time-steps number (TR), allowing ML strategies such as DNNs to get applied. This is merely necessary owing to data utilizing from distinct workloads in which the IFA does not require it, resampling isn't necessary as a strategy. The SciPy package implements the Fourier resampling algorithm. This approach is working using the Nyquist-Shannon sampling theorem. The underlying principle is that altering a signal's frequency domain spectrum changes the number of samples required to collect all trace information. Empirical studies show that the best results are obtained while the goal samples number equals the average number of time-steps for every workload. The resampled traces are fed into a multi-class classifier that trained for identifying the microarchitectural unit if an issue in performance is present.

4. RESULT AND DISCUSSION

This experimental system was created utilizing the Contiki Operating System and a software defined network solution for wireless sensor networks (SDN-WISE). The experiment is conducted out utilizing the virtual box platform, which is embedded in the Windows 10 OS. Ubuntu OS has been installed on a virtual box, together with an open-source controller and switch. This work focuses on the bug localization performance in cores. Sections IV-A to IV-C detail the experimental setup features, while section IV-D describes the alterations performed to adapt the proposed methodology in the memory system's performance bug localization setup. There are 190 SimPoints from ten programs retrieved in the SPEC CPU2006 benchmark and used it as a workloads set to assess the proposed DNN-IFA model with the Tensorflow library. Similarly, the packets of UDP, ICMP and TCP packets are produced by trustworthy users as well as attackers by scapy and a Python script.

The design of network topology by mininet for SDN switches with single controller and 40 devices are linked to detect DDoS attack is shown in Figure 2. The connected devices in network include wireless equipment with several switches and host that connect in the network. For experimental purpose, the attack host is considered to four different hosts that get connected with a single switch. However, the fake request sends by intruder continuously from hosts. Hence, the performance of detecting DDoS attack is measured by precision (False positive rate), and recall (True positive rate). Some bug types have no variants utilized for training the DNN-IFA, DNN-FA and DNN models that allowing the approaches to be tested on bugs it has not encountered previously, which represented as "unseen bug types". For the other bug types, certain permutations have been employed to train the ML models, whereas others are only tested. The results discuss about model performs over bugs which are comparable but not identical to those it encountered previously as well as when a similar bug observed in impact designs considered. Table 1 illustrates the bug localization classification method with confusion matrix metrics score to determine the performance of bug detection.

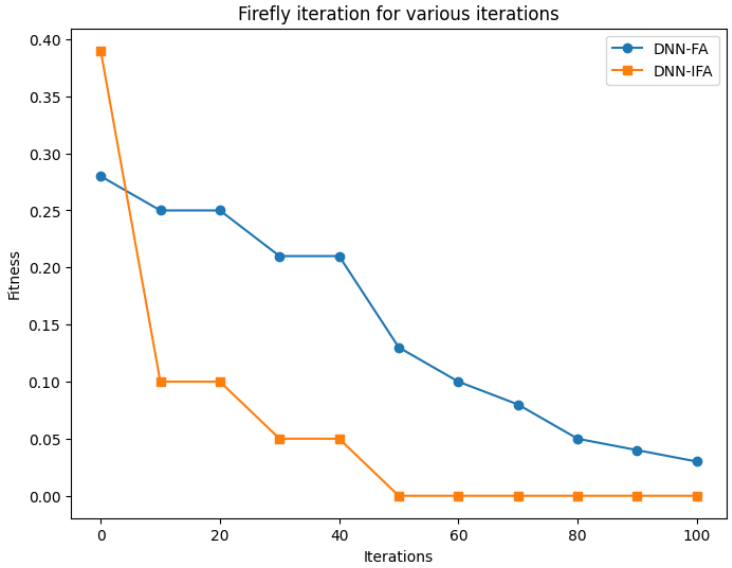

Figure 2. Comparison of Firefly Fitness

Table 1. Confusion matrix for bug localization classification method

Bug Localization classification method	Accuracy (%)	TPR	FPR
DNN	93.14	0.973	0.027
DNN-FA	95.79	0.986	0.014
DNN-IFA	98.97	0.998	0.002

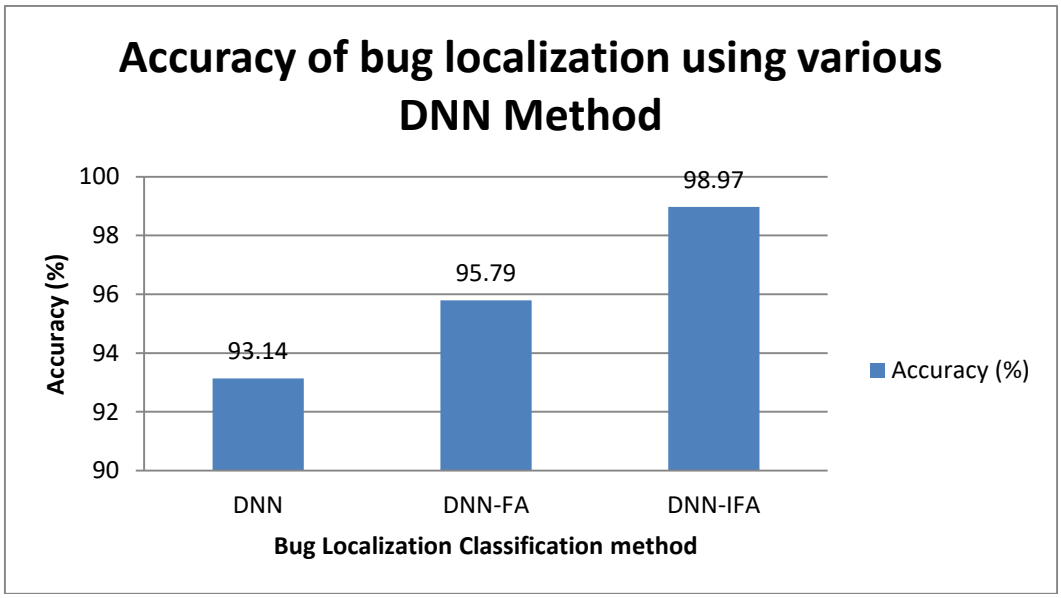

Figure 3. Performance accuracy in bug localization using various DNN methods

Figure 3 illustrates the accuracy of bug localization performance for various DNN methods in which DNN-IFA has better detection of bug existence accuracy as 98.79% which is comparatively higher than DNN-FA and DNN methods.

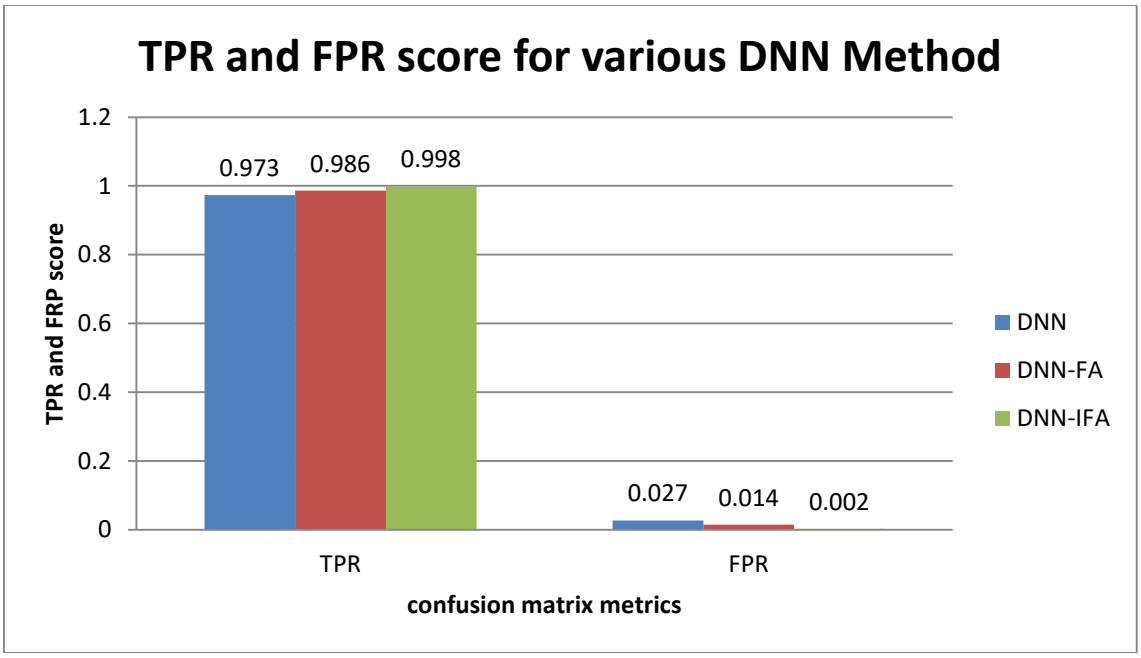

Figure 4. TPR and FPR performance of bug localization using various DNN methods

Figure 4 illustrates the TPR and FPR for bug localization with proposed DNN-IFA method in which TPR with high score as 0.998 and FPR with very low score as 0.002 which is comparatively better than DNN-FA and DNN methods. Thus, the automated bug localization is better with DNN-IFA method for wireless network with several hosts.

5. CONCLUSION

This study introduces IFA and DNN as two automated approaches for recognizing microprocessor performance bugs using ML method. This technique has generated data from legacy designs in determining the links among performance counter behavior and performance bug sites. This research approach has generated more accurate with even large data due to ensemble of two approaches as DNN-IFA and made evaluated by discovering additional accuracy improvement. Simulation findings for the analyzed bugs reveal 98.97% for detecting bugs existence and with least FPR as 0.002. The ML models collect knowledge from historical designs while avoiding the old approaches of reference performance models are error-prone to create. The simulation findings suggest that the DNN-IFA analysis can detect bug existence with 0.998 TPR in a newly developed microarchitecture. With this work, it intends to raise the research community's emphasis to the broad performance validation domain.

REFERENCES

[1]. Freeman, R.; Kawa, J.; Singhal, K. Synopsys' Journey to Enable TCAD and EDA Tools for Superconducting Electronics. 2020.
[2]. J. D. McCalpin, "HPL and DGEMM performance variability on the Xeon platinum 8160 processor," in International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 225–237.
[3]. T. Li, H. Zou, D. Luo, and W. Qu, "Symbolic simulation enhanced coverage-directed fuzz testing of rtl design," in 2021 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2021, pp. 1–5.
[4]. K. Ganesan, F. Lonsing, S. S. Nuthakki, E. Singh, M. R. Fadiheh, W. Kunz, D. Stoffel, C. Barrett, and S. Mitra, "Effective pre-silicon verification of processor cores by breaking the bounds of symbolic quick error detection," arXiv preprint arXiv:2106.10392, 2021.
[5]. H. Liang, L. Sun, M. Wang, and Y. Yang, ''Deep learning with customized abstract syntax tree for bug localization,'' IEEE Access, vol. 7, pp. 116309–116320, 2019.
[6]. W. Zhang, Z. Li, Q. Wang, and J. Li, ''FineLocator: A novel approach to method-level fine-grained bug localization by query expansion,'' Inf. Softw. Technol., vol. 110, pp. 121–135, Jun. 2019.
[7]. H. Xuan, T. Ferdian, L. Ming, L. David, and S. Shu-Ting, ''Deep transfer bug localization,'' IEEE Trans. Softw. Eng., vol. 47, no. 7, pp. 1368–1380, Jul. 2021.

[8]. E. Carvajal Barboza, S. Jacob, M. Ketkar, M. Kishinevsky, P. Gratz, and J. Hu, "Automatic microprocessor performance bug detection," in IEEE International Symposium on High-Performance Computer Architecture, 2021, pp. 545–556.
[9]. Y. Wang, Y. Yao, H. Tong, X. Huo, M. Li, F. Xu, and J. Lu, ''Enhancing supervised bug localization with metadata and stack-trace,'' Knowl. Inf. Syst., vol. 62, no. 6, pp. 2461–2484, Jun. 2020.
[10]. S. Cheng, X. Yan, and A. A. Khan, ''A similarity integration method based information retrieval and word embedding in bug localization,'' in Proc. IEEE 20th Int. Conf. Softw. Qual., Rel. Secur. (QRS), Dec. 2020, pp. 180–187.
[11]. H. Liang, L. Sun, M. Wang, and Y. Yang, ''Deep learning with customized abstract syntax tree for bug localization,'' IEEE Access, vol. 7, pp. 116309–116320, 2019.
[12]. A. Ciborowska, M. J. Decker, and K. Damevski, ''Online adaptable bug localization for rapidly evolving software,'' 2022, arXiv:2203.03544.
[13]. F. Fang, J. Wu, Y. Li, X. Ye, W. Aljedaani, and M. W. Mkaouer, ''On the classification of bug reports to improve bug localization,'' Soft Comput., vol. 25, no. 11, pp. 7307–7323, Jun. 2021.
[14]. S. Khatiwada, M. Tushev, and A. Mahmoud, ''On combining IR methods to improve bug localization,'' in Proc. 28th Int. Conf. Program Comprehension, Jul. 2020, pp. 252–262.
[15]. A. Razzaq, J. Buckley, J. V. Patten, M. Chochlov, and A. R. Sai, ''Boost- NSift: A query boosting and code sifting technique for method level bug localization,'' in Proc. IEEE 21st Int. Work. Conf. Source Code Anal.Manipulation (SCAM), Sep. 2021, pp. 81–91.
[16]. F. Solt, B. Gras, and K. Razavi, "Cellift: Leveraging cells for scalable and precise dynamic information flow tracking in rtl," in 31st USENIXSecurity Symposium (USENIX Security 22), 2022, pp. 2549–2566.
[17]. J. Balkind, K. Lim, F. Gao, J. Tu, D. Wentzlaff, M. Schaffner, F. Zaruba, and L. Benini, "Openpiton+ ariane: The first open-source, smp linuxbooting risc-v system scaling from one to many cores," in Workshop on Computer Architecture Research with RISC-V (CARRV), 2019, pp. 1–6.
[18]. Dinghao Liu, QiushiWu, Shouling Ji, Kangjie Lu, Zhenguang Liu, Jianhai Chen, and Qinming He. Detecting missed security operations through differential checking of object-based similar paths. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pages 1627–1644, 2021.
[19]. V. Murali, L. Gross, R. Qian, and S. Chandra, ''Industry-scale IR-based bug localization: A perspective from Facebook,'' in Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP), May 2021, pp. 188–197.
[20]. A. Takahashi, N. Sae-Lim, S. Hayashi, and M. Saeki, ''An extensive study on smell-aware bug localization,'' J. Syst. Softw., vol. 178, Aug. 2021, Art. no. 110986.
[21]. S. Sangle, S. Muvva, S. Chimalakonda, K. Ponnalagu, and V. Gopalan Venkoparao, ''DRAST—Adeep learning and AST based approach for bug localization,'' 2020, arXiv:2011.03449.