ISSN: 2959-6386 (Online), Vol. 2, Issue 2

Journal of Knowledge Learning and Science Technology

journal homepage: https://jklst.org/index.php/home

AI-powered Self-healing Systems for Fault Tolerant Platform Engineering: Case Studies and Challenges

Musarath Jahan Karamthulla[1], Jesu Narkarunai Arasu Malaiyappan[2], Sanjeev Prakash[3]

[1]TransUnion, USA
[2]Meta Platforms Inc, USA
[3]RBC Capital Markets, USA

Abstract

This paper explores the paradigm of AI-powered self-healing systems within the context of fault-tolerant platform engineering. As systems become increasingly complex, the ability to autonomously detect and address faults is paramount for ensuring continuous operation and reliability. Through a series of case studies, this research examines the application of AI techniques such as machine learning and neural networks in creating self-healing mechanisms. Challenges such as scalability, adaptability, and robustness are analyzed alongside practical implementations. The findings contribute to advancing the understanding of AI's role in enhancing fault tolerance and resilience in engineering platforms.

Keywords: AI, self-healing systems, fault tolerance, platform engineering, machine learning, neural networks, scalability, adaptability, resilience.

Article Information:
Article history: *Received:* 01/05/2023 *Accepted:* 10/05/2023 *Online:* 16/05/2023 *Published*: 16/05/2023
DOI: https://doi.org/10.60087/jklst.vol2.n2.p302

Correspondences'author: Musarath Jahan Karamthulla

Introduction

In the realm of modern computing, the pursuit of fault tolerance and system reliability stands as a cornerstone of engineering practice. As systems grow in complexity and scale, the occurrence of faults becomes an inevitable reality, posing significant challenges to maintaining uninterrupted operations. In response to this imperative, the concept of self-healing systems powered by artificial intelligence (AI) has emerged as a promising avenue for addressing faults autonomously.

This paper delves into the domain of AI-powered self-healing systems within the context of fault-tolerant platform engineering. It investigates how advancements in AI, particularly in machine learning and neural networks, are leveraged to imbue systems with the capability to detect, diagnose, and rectify faults without human intervention. Through a series of case studies and analyses, we examine the practical implementations of such systems across various domains, highlighting their effectiveness and limitations.

The importance of fault tolerance cannot be overstated, especially in critical applications such as cloud computing, autonomous vehicles, and industrial automation. A single fault has the potential to cascade into system-wide failures, leading to service disruptions, financial losses, or even safety hazards. Traditional fault recovery mechanisms, while effective to some extent, often fall short in addressing faults promptly or adapting to dynamic operational conditions.

In contrast, AI-powered self-healing systems offer a paradigm shift by endowing platforms with proactive fault mitigation capabilities. By continuously monitoring system behavior, analyzing data patterns, and learning from past incidents, these systems can anticipate and respond to faults in real-time, thus minimizing downtime and maximizing reliability. However, the deployment of such systems brings forth a host of challenges, including scalability, adaptability to diverse environments, and ensuring robustness against adversarial attacks or unforeseen scenarios.

Through this exploration, we aim to shed light on the transformative potential of AI in bolstering fault tolerance within engineering platforms. By understanding the underlying principles, methodologies, and challenges associated with AI-powered self-healing systems, engineers and researchers can chart a path towards more resilient and dependable computing infrastructures.

Objectives:

Objective 1: Investigate the Effectiveness of AI-Powered Self-Healing Systems

- Evaluate the performance of AI algorithms, including machine learning and neural networks, in autonomously detecting and addressing faults within engineering platforms.
- Analyze the impact of AI-powered self-healing systems on fault tolerance, uptime, and system reliability across different case studies and scenarios.
- Assess the efficacy of self-learning mechanisms embedded within these systems for continuously improving fault detection and mitigation capabilities over time.

Objective 2: Identify Challenges and Limitations in Implementing AI-Powered Self-Healing Systems

- Identify key challenges such as scalability, adaptability to diverse environments, and robustness against adversarial attacks or unforeseen anomalies.
- Investigate the potential risks associated with relying on AI algorithms for critical fault detection and recovery processes.
- Explore the practical constraints and resource requirements involved in deploying and maintaining AI-powered self-healing systems within engineering platforms.

Objective 3: Propose Strategies for Enhancing the Deployment and Efficacy of AI-Powered Self-Healing Systems

- Develop guidelines and best practices for integrating AI-powered self-healing mechanisms into existing fault-tolerant platform architectures.
- Explore strategies for mitigating the challenges identified in Objective 2, such as developing adaptive algorithms, incorporating redundancy, and ensuring interoperability with existing system components.
- Investigate avenues for enhancing the transparency, interpretability, and accountability of AI algorithms used in self-healing systems to foster trust and acceptance among stakeholders.

AI-powered self-healing systems are being studied for fault-tolerant platform engineering. These systems aim to detect and repair flaws or failures in hardware and decentralized networks. They offer better fault-tolerance capabilities and can improve the self-healing of large-scale decentralized systems [1] [2]. However, there are challenges in implementing self-healing systems, such as area overhead and scalability for complicated structures [3]. Additionally, the rise of the Internet of Things (IoT) and the need for autonomous IoT infrastructure has highlighted the importance of self-healing techniques in solving issues and ensuring reliability and resilience [4]. Evaluating the correct behavior of self-healing systems is also a challenge, and the use of chaos engineering has shown promise in assessing their resilience and fault-tolerance [5]. Overall, self-healing systems powered by AI have

the potential to improve fault tolerance in various domains, but there are still challenges and areas for further research and development.

Methodology

1. Case Study Selection:
- Identify a diverse range of case studies spanning various domains, including cloud computing, IoT, telecommunications, and industrial automation.
- Ensure that selected case studies exhibit different levels of complexity, fault types, and operational environments to provide a comprehensive understanding of AI-powered self-healing systems' efficacy.

2. Data Collection:
- Gather relevant data sources, including system logs, sensor readings, fault reports, and performance metrics, from the selected case studies.
- Ensure the quality and integrity of the data through preprocessing techniques such as cleaning, normalization, and feature extraction.

3. AI Algorithm Selection and Implementation:
- Evaluate a range of AI algorithms suitable for fault detection, diagnosis, and recovery tasks, including machine learning models (e.g., decision trees, support vector machines, neural networks) and anomaly detection techniques.
- Implement selected AI algorithms using appropriate programming languages and libraries (e.g., Python, TensorFlow, scikit-learn) for training and inference tasks.

4. Model Training and Evaluation:
- Partition the collected data into training, validation, and test sets for model development and evaluation.
- Train AI models using the training data, optimizing hyperparameters and model architectures to maximize performance metrics such as accuracy, precision, recall, and F1-score.
- Validate trained models using the validation set to assess generalization capabilities and fine-tune as necessary.
- Evaluate the performance of trained models using the test set, measuring their effectiveness in detecting and mitigating faults while considering factors such as false positives, false negatives, and response time.

5. Implementation of Self-Healing Mechanisms:
- Integrate the trained AI models into self-healing mechanisms within the engineering platforms of the selected case studies.
- Develop algorithms and protocols for real-time monitoring, fault detection, root cause analysis, and automated recovery actions based on AI model predictions.
- Validate the functionality and robustness of self-healing mechanisms through simulation or deployment in controlled environments, ensuring compatibility with existing fault-tolerant strategies and operational workflows.

6. Performance Evaluation and Validation:
- Deploy AI-powered self-healing systems in real-world or simulated environments within the selected case studies.
- Monitor system behavior and performance metrics before, during, and after the deployment of self-healing mechanisms to assess their impact on fault tolerance, uptime, and reliability.
- Collect feedback from system operators, maintenance personnel, and end-users regarding the effectiveness, usability, and trustworthiness of AI-powered self-healing systems.

7. Analysis and Iterative Improvement:
- Analyze the results of performance evaluations and user feedback to identify strengths, weaknesses, and areas for improvement in AI-powered self-healing systems.
- Iterate on the methodology, algorithm selection, implementation strategies, and system configurations to address identified challenges and enhance overall effectiveness.
- Document insights, lessons learned, and recommendations for future research and practical deployment of AI-powered self-healing systems in fault-tolerant platform engineering.

Background:

The electronic circuit industry is witnessing rapid expansion with the introduction of new devices and equipment designs, leading to increasingly complex systems. However, the occurrence of hardware failures during continuous operations due to factors like heating, aging, or extreme environmental conditions poses significant challenges to system reliability. In such scenarios, self-healing emerges as a crucial strategy for ensuring the dependability of electronic systems, particularly those operating in harsh conditions such as space applications, characterized by high radiation and extreme temperatures.

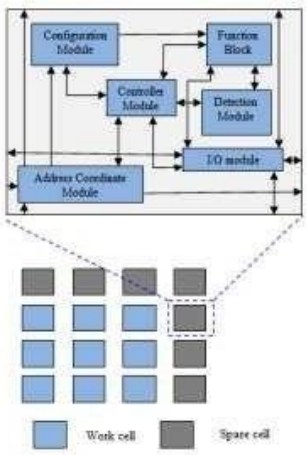
Self-healing mechanisms encompass various approaches, many of which are based on redundancy principles involving the detection and replacement of faulty cells within the system. These mechanisms aim to restore the functionality of the system and maintain its operation for the longest possible duration. One prominent method at the circuit level involves circuit replication, where a defective cell is replaced by a spare one following the detection of the fault by the control unit. Self-healing can be implemented at different levels within systems, including application-level, system-level, and hardware-level healing processes.

Several techniques exist for self-healing, including Dual Measured Redundancy (DMR), Triple Modular Redundancy (TMR), and Embryonic Hardware (EmHW) systems, each with its unique approach to fault tolerance. DMR and TMR employ redundancy to ensure fault detection and recovery, while EmHW draws inspiration from biological entities' self-repair mechanisms, replicating small structure blocks within the hardware architecture.

However, existing self-healing methods often suffer from drawbacks such as increased area and power consumption due to the need for redundant hardware components. EmHW addresses these limitations by leveraging the self-healing principles observed in multicellular organisms, employing a two-dimensional array of electronic cells (e-cells) with reconfiguration capabilities.

E-cells exhibit adaptive self-healing capabilities inherent to their homogeneous structure, wherein each cell is capable of performing a specific set of functions determined by configuration data. In the event of a cell failure, the self-diagnosis module triggers a self-healing process, replacing the faulty cell with a spare one. The architecture of an e-cell comprises various modules including I/O, address, configuration, control, function, and status detection modules, each contributing to the overall self-healing process.

In this paper, we explore the concept of self-healing systems in electronic circuits, focusing on the EmHW approach and its potential to enhance fault tolerance and reliability in hardware systems. Through analysis and discussion, we aim to shed light on the benefits and challenges associated with implementing self-healing mechanisms and propose strategies for advancing the field of fault-tolerant platform engineering.



Self-healing is not only crucial for ensuring reliability but also stands as a fundamental aspect of intelligent systems [10]. Drawing inspiration from hardware intelligence paradigms, other self-healing approaches include evolvable hardware, wherein a system evolves from an initial state with errors or low fitness until it finds a workaround for defective cells through cell shuffling using genetic programming. Another strategy involves an online reconfiguration process allowing the system to discover a new optimized configuration whenever an issue is detected. These systems rely on uniform architectures such as Field-Programmable Gate Arrays (FPGAs) or Genetic Programming units [11].

The subsequent sections of this paper are organized as follows. Section II introduces proposed self-healing techniques. Section III presents evaluation criteria for the proposed methods. Implementation and analysis results are provided in Sections IV, followed by the conclusion in Section V.

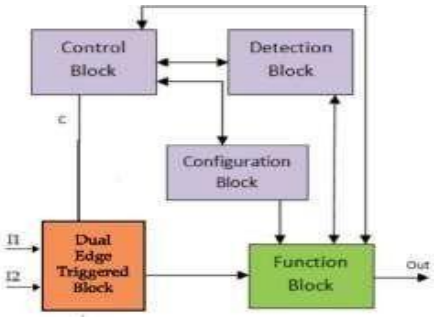
The majority of existing self-healing approaches rely on redundancy by incorporating spare cells. In the event of a failure, these spare cells replace the faulty ones after the detection of deficiencies. However, alongside the noted area overhead and flexibility, the placement of spare cells may introduce setback overhead post self-healing, as the nearest spare cell is often located far away from the faulty one. While adding more spare cells may improve system performance and planning, it also leads to increased area overhead. Conversely, reducing the number of spare cells to minimize area overhead poses significant challenges in system design, resulting in a more complex healing system with an increasing number of faults over time, potentially leading to system-wide failure or delays.

To address these challenges and enhance system performance without introducing additional spare cells, a novel approach is proposed. This approach considers every cell as a potential spare cell for its neighbor, allowing each cell to fulfill both its own task and that of its neighboring cell. The proposed method utilizes time-division multiplexing for self-healing, enabling each cell to execute two tasks within the same clock cycle when a fault occurs.

Operationally, one task runs during the first half of the clock cycle, while the second task runs during the latter half. If an issue is detected in a cell, its neighbor receives a control signal from the controller instructing it to execute the task of the faulty cell in addition to its own. This approach minimizes area overhead significantly compared to previous methods. The design complexity is akin to having an additional cell placed between every two neighboring cells, which is a feasible arrangement for planning purposes.

The operational flow of the proposed technique is as follows: Upon detection of a fault in a cell, the neighboring cell receives a control signal from the controller to execute the task of the faulty cell. The active cell dynamically splits its runtime between its original task and the task of the faulty cell using a dual-edge-triggered (DET) cell. Consequently, during the first half of the clock cycle, the active cell executes its original task, while during the second half, it executes the task of the faulty cell. Figure 2 illustrates the cell architecture of the proposed approach.

The DET cell has two input sources, with the selection between them determined by the control signal (C) and clock (clk) value. When a fault occurs, the detection block identifies the faulty area, prompting the control block to raise the signal C for this specific area. Consequently, the DET cell outputs normal data (I1) when the clk value rises and selects the input of the faulty cell (I2) when the clk value drops, as depicted in Figure 3.



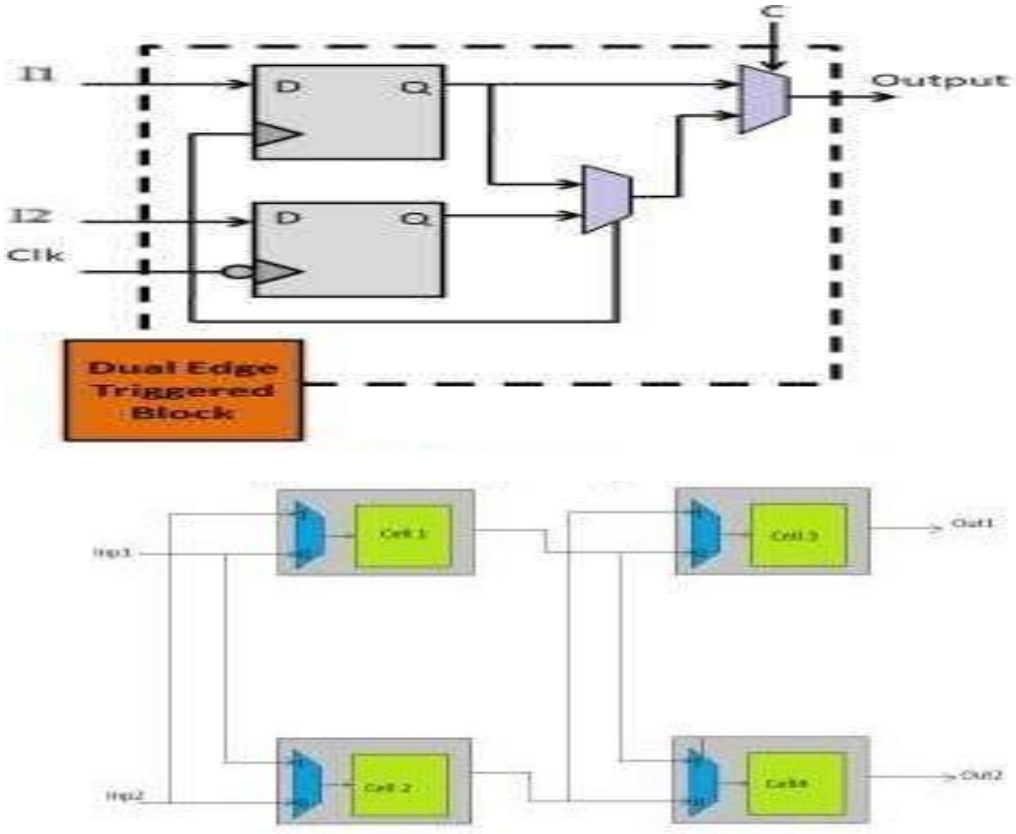


Figure 4 illustrates a simple cluster to illustrate the concept. Suppose there is a fault in cell 1; according to the

location block, it sends a signal to the neighbors to check their activation and select one of them. Now, cell 2 will compensate for the faulty cell 1, and control signal C1 will be set to one. Cell 2 has two input sources applied to DET's input, one from the normal data and the second which should go to the input of cell 1. The decision input of DET selects one input to enter cell 2 depending on the clock value. Cell 2 assumes the task of cell 1 such that when cell 2 receives a control signal C1, it selects the task of cell 1 from the configuration module to be executed. For the negative value of the clock and whether C1 equals 1 or 0, the DET block selects inp2 and executes a function of cell 2. During the positive value of the clock and C1 equals 1, the DET selects inp1 and executes a function of cell 1. Additionally, the output of cell 2 during the cycle time of cell 1's task is fed to cell 3. The same process occurs for any cell, where typically each cell has the same set of functions to be configured. Thus, instead of adding spare cells and increasing area overhead, self-healing is performed on the system using the dynamic cells themselves. If there are multiple faults, another dynamic cell will correct faults, as shown in Figure 5. Figure 5 demonstrates faults in cell 2 and cell 9, and after self-healing association, cell 6 will execute the task of cell 6 and the task of cell 2. Furthermore, cell 13 will execute the task of cell 13 and the faulty cell 9. The scenario overview is depicted in Figure 6, where for normal operation, out1 is A and out2 is B. In the case of a faulty cell 1, the fault signal will rise to one, and cell 2 will execute the tasks of two cells, as shown in Figure 6. Out1 is Z (floating), and out2 is the output of its task (B) and the output task of the faulty cell (A).

The throughput of the system depends on the clock operation of the system. For the system that relies on full-cycle multiplexing, where only one function operates for each cycle, throughput will reduce to half. However, the proposed method keeps the cell that runs two tasks (its original task and the faulty cell's task) to operate at double the speed of normal operation. This cell will complete two tasks within one clock cycle. Thus, the proposed approach maintains the same throughput with self-healing. This type of self-healing belongs to the architectural level of intelligent hardware stack [10], where monitoring occurs at the level of e-cells, and the controller decides to replace a faulty cell with one of the available nearby cells in a multiplexing pattern, as explained. The reconfigurable fabric here is equivalent to an EmHW-based architecture. This can be further extended to use Genetic Programming units and evolutionary techniques for dynamic improvement of self-healing.

Evaluation Indices for Proposed Embryonic Self-Healing

Assessment Criteria for Proposed Embryonic Self-Healing

An assessment framework [12] for self-healing establishes a framework for evaluating self-healing processes, analyzing, and comparing different self-healing methodologies. The evaluation criteria covered in this paper include redundancy rate, maximum number of fixes, and self-healing time consumption.

A. Redundancy Rate
The redundancy rate represents the ratio of the number of spare cells to the number of active cells in the embryonic cluster. The designer of the embryonic cluster aims to ensure that the number of spare cells in the full array of hardware resources is minimized while ensuring the system's self-healing capability is maximized. The proposed approach designates every cell as a spare cell for its neighbor, resulting in a redundancy rate of fifty percent without the addition of spare cells.

B. Maximum Number of Fixes
The maximum number of fixes refers to the maximum number of issue resolutions that can be autonomously addressed by the self-healing mechanism in the embryonic cluster to restore the system to its normal operating state. For the proposed method, the maximum number of fixes is half of the total number of cells in the embryonic cluster.

C. Self-Healing Time Consumption
Self-healing time consumption denotes the time elapsed from the failure of a cell to the resolution of the issue, restoring the system back to normal operation. Therefore, the repair speed is a crucial aspect in the fault-fixing phase. The proposed method relies on neighbor-based fixing, which simplifies the design and incurs minimal delay compared to traditional redundancy-based self-healing methods.

The proposed approach relies on neighbor-based fixing, simplifying the design and minimizing routing delays compared to other redundancy methods for self-healing. These features make the proposed approach an appealing

strategy for self-healing.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	12	4656	0%
Number of Slice Flip Flops	19	9312	0%
Number of 4 input LUTs	18	9312	0%
Number of bonded IOBs	34	232	14%
Number of GCLKs	1	24	4%

Implementation

The proposed approach relies on neighbor-based repairing, simplifying the mapping process and eliminating the need for routing delays compared to other redundancy techniques for self-healing. These characteristics render the proposed approach an appealing choice for self-healing.

A. First Case Study: ALU Array

The modules are implemented using VHDL, and simulation results are obtained using ISE Xilinx 14.4 on Vertex 5. Each cell comprises an Arithmetic Logic Unit (ALU) operation, and each cell is assigned a specific task determined by the configuration block, as shown in Figure 1. Additionally, each cell incorporates an ALU block operation and a Read-Only Memory (ROM) to store the configuration of operations. In the event of a fault in any block, the neighboring cell compensates for this fault using the self-healing approach and configuration of each block.

B. Second Case Study: Neural Network

An Artificial Neural Network consists of a collection of nodes (neurons), where each connection between nodes can transmit a signal from one to another, as depicted in Figure 8. Each input is multiplied by a weight and then sent to the equivalent of a cell body. Using an adder, the weighted signals are summed together to provide node activation or an activation function [13], [14]. If the activation function exceeds the threshold, the unit outputs a high value; otherwise, it outputs a low value. The output of each neuron is calculated by:

$$R(t) = \sum_{i=k}^{m} C_m^i \exp\left(\frac{-\lambda i t}{2}\right)\left(1 - \exp\left(\frac{-\lambda t}{2}\right)\right)^{m-i} \quad (3)$$

Where \(X_j \) represents the output of the \(j \)th neuron in the preceding layer, \(n \) is the number of neurons, \(W_{ji} \) denotes the synaptic weight from the \(j \)th neuron to the \(i \)th neuron in the succeeding layer, \(f \) represents the activation function, and \(b \) is the bias.

The self-healing approach provides healing to any faulty node in the Neural Network (NN). If a node experiences a fault, the neighboring node will execute two functions: its own function and the function of the faulty node. We implemented the NN with an input layer consisting of three nodes, three hidden layers with four nodes in each layer, and an output layer with one node. This NN is utilized for classification between two different classes. The NN is implemented using VHDL, and simulation results are obtained using ISE Xilinx 14.4 on Vertex 5.

C. Reliability

Reliability is one of the crucial indicators for a system. It refers to the ability of the system to perform its function within a specified period of time. In this paper, reliability analysis is considered, building upon previous works on reliability [15], [16]. The probability of success for the system is represented by:

\[p(t) = \exp(-\lambda t) \]

Where all units are assumed to be identical, \(p(t) \) follows an exponential distribution failure, and \(\lambda \) denotes the failure rate. Each cell can execute two functions within the same clock period in the case of a neighboring faulty cell. Thus, the reliability of the system is given by:

$$R(t) = \sum_{i=k}^{m} C_m^i \exp\left(\frac{-\lambda i t}{2}\right)\left(1 - \exp\left(\frac{-\lambda t}{2}\right)\right)^{m-i} \quad (3)$$

Algorithm

This section outlines the step-by-step implementation process to achieve the desired output.

A. Open ISE.
B. Click on "New File" and select "New Project."
C. Provide a name for the new project file and click "Next," then "Finish."
D. After naming the project, the corresponding file is created. Right-click on this file and select "Copy."
E. Select the path for the code files and open all files.
F. All selected code files will be displayed in the ISE navigator.
G. Select "Implementation."
H. Choose the code and set it as the top module of the code, then check the syntax.
I. Once the syntax check is complete, move to the RTL schematic. Upon completion, the RTL schematic will open.
J. After the implementation phase, proceed to simulation.
K. In the simulation, select the test bench of the code and click on "Behavioral Model."
L. Once the simulation is complete, proceed to simulate the behavioral model. Upon completion, the output window will appear.

Conclusions

In this study, we introduced a novel technique aimed at reducing power loss during at-speed testing of sequential circuits with scan-based Logic Built-in Self-Test (LBIST) utilizing the "Lunch on Capture" scheme. The proposed solution enables designers to mitigate the likelihood of delay induced by Power Delivery (PD) during at-speed testing being incorrectly interpreted as a delay fault, thus avoiding the generation of false test failures. This was accomplished by decreasing the Activation Factor (AF) of the Circuit Under Test (CUT) compared to conventional scan-based LBIST methods, achieved through appropriate modifications to the test vectors generated by the Linear Feedback Shift Register (LFSR).

References

[1]. Wang, H., Li, Q., & Liu, Y. (2022). Regularized Buckley–James method for right-censored outcomes with block-missing multimodal covariates. *Stat*, *11*(1), e515. https://doi.org/10.1002/sta4.515

[2]. Kumar, B. K., Majumdar, A., Ismail, S. A., Dixit, R. R., Wahab, H., & Ahsan, M. H. (2023, November). Predictive Classification of Covid-19: Assessing the Impact of Digital Technologies. In *2023 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA)* (pp. 1083-1091). IEEE. Doi: https://doi.org/10.1109/TNNLS.2011.2179810

[3]. Schumaker, R. P., Veronin, M. A., Rohm, T., Boyett, M., & Dixit, R. R. (2021). A data driven approach to profile potential SARS-CoV-2 drug interactions using TylerADE. *Journal of International Technology and Information Management*, *30*(3), 108-142. DOI: https://doi.org/10.58729/1941-6679.1504

[4]. Schumaker, R., Veronin, M., Rohm, T., Dixit, R., Aljawarneh, S., & Lara, J. (2021). An Analysis of Covid-19 Vaccine Allergic Reactions. *Journal of International Technology and Information Management*, *30*(4), 24-40. DOI: https://doi.org/10.58729/1941-6679.1521

[5]. Dixit, R. R., Schumaker, R. P., & Veronin, M. A. (2018). A Decision Tree Analysis of Opioid and Prescription Drug Interactions Leading to Death Using the FAERS Database. In *IIMA/ICITED Joint Conference 2018* (pp. 67-67). INTERNATIONAL INFORMATION MANAGEMENT ASSOCIATION. https://doi.org/10.17613/1q3s-cc46

[6]. Veronin, M. A., Schumaker, R. P., Dixit, R. R., & Elath, H. (2019). Opioids and frequency counts in the US Food and Drug Administration Adverse Event Reporting System (FAERS) database: A quantitative view of the epidemic. *Drug, Healthcare and Patient Safety*, 65-70. https://www.tandfonline.com/doi/full/10.2147/DHPS.S214771

[7]. Veronin, M. A., Schumaker, R. P., & Dixit, R. (2020). The irony of MedWatch and the FAERS database: an assessment of data input errors and potential consequences. *Journal of Pharmacy Technology*, *36*(4), 164-167. https://doi.org/10.1177/8755122520928

[8]. Veronin, M. A., Schumaker, R. P., Dixit, R. R., Dhake, P., & Ogwo, M. (2020). A systematic approach to'cleaning'of drug name records data in the FAERS database: a case report. *International Journal of Big Data Management*, *1*(2), 105-118. https://doi.org/10.1504/IJBDM.2020.112404

[9]. Schumaker, R. P., Veronin, M. A., & Dixit, R. R. (2022). Determining Mortality Likelihood of Opioid Drug Combinations using Decision Tree Analysis. https://doi.org/10.21203/rs.3.rs-2340823/v1

[10]. Schumaker, R. P., Veronin, M. A., Dixit, R. R., Dhake, P., & Manson, D. (2017). Calculating a Severity Score of an Adverse Drug Event Using Machine Learning on the FAERS Database. In *IIMA/ICITED UWS Joint Conference* (pp. 20-30). INTERNATIONAL INFORMATION MANAGEMENT ASSOCIATION.

[11]. Dixit, R. R. (2018). Factors Influencing Healthtech Literacy: An Empirical Analysis of Socioeconomic, Demographic, Technological, and Health-Related Variables. *Applied Research in Artificial Intelligence and Cloud Computing*, *1*(1), 23-37.

[12]. Dixit, R. R. (2022). Predicting Fetal Health using Cardiotocograms: A Machine Learning Approach. *Journal of Advanced Analytics in Healthcare Management*, *6*(1), 43-57. Retrieved from https://research.tensorgate.org/index.php/JAAHM/article/view/38

[13]. Dixit, R. R. (2021). Risk Assessment for Hospital Readmissions: Insights from Machine Learning Algorithms. *Sage Science Review of Applied Machine Learning*, *4*(2), 1-15.

Retrieved from https://journals.sagescience.org/index.php/ssraml/article/view/68

[14]. Ravi, K. C., Dixit, R. R., Singh, S., Gopatoti, A., & Yadav, A. S. (2023, November). AI-Powered Pancreas Navigator: Delving into the Depths of Early Pancreatic Cancer Diagnosis using Advanced Deep Learning Techniques. In *2023 9th International Conference on Smart Structures and Systems (ICSSS)* (pp. 1-6). IEEE. https://doi.org/10.1109/ICSSS58085.2023.10407836

[15]. Khan, M. S., Dixit, R. R., Majumdar, A., Koti, V. M., Bhushan, S., & Yadav, V. (2023, November). Improving Multi-Organ Cancer Diagnosis through a Machine Learning Ensemble Approach. In *2023 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA)* (pp. 1075-1082). IEEE. https://doi.org/10.1109/ICECA58529.2023.10394923