[1] Devesh Lowe*
[2] Dr Mithilesh Kumar Dwivedi

Enhanced Data Race Detection Through Dynamic Control Flow Analysis for Aspect-Oriented Program Using Ss-Lsgru and G-Csoa

Journal of Electrical Systems

Abstract: - Aspect Oriented Programming (AOP) language is a high-level object-oriented language, which is widely used for generating web applications. As AOP is designed in a multi-threaded manner, the data race occurs in a program when different threads access the shared memory resource. But, none of the existing works handle the dynamic control flow of AOP, resulting in higher levels of false positives and false negatives in Data Race Detection (DRD). Therefore, in this work, an efficient framework is proposed for detecting the data race of Aspect Oriented Programming (AOP) language using Eisen Cosine Correlation distance based Entropy Variance KMeans (ECC-EVKMeans), SoftSwish – Linear Scaling Gated Recurrent Unit (SS-LSGRU), and Kullback Leibler -based Fuzzy Bayesian Inference System (KL-FBIS). Primarily, the proposed system acquires an AOP of various applications; then, its number of variables along with the methods are extracted. Further, the context-sensitive information of code is analyzed in the thread analysis phase using the control flow graph. Subsequently, the dynamic scope of pointers is evaluated in escape analysis, and the single parameterized analysis of each method is attained by the compositional pointers. Meanwhile, the test cases are generated, and the significant test cases are selected and clustered using G-CSOA and ECC-EVKMeans, respectively. Then, by using the BERT algorithm, the vector values of corresponding words in test cases are returned. Further, the vector values are separated using the KL-FBIS approach to minimize the nested loops. Eventually, the vector values are trained using the SS-LSGRU classifier and also tested with real-time AOP for detecting the race condition. The experimental results show that the proposed system detects data race with 98.26% accuracy and 98.95% precision in 37751ms. Also, the important test cases are selected with 97.85% fitness by using the proposed technique.

Keywords: Bidirectional Encoders Representations from Transformers (BERT), Galois - Chameleon Swarm Optimization Algorithm (G-CSOA), Test case, loops, Threads, Word vectors, Data race.

1. INTRODUCTION

In the advanced technology, many applications and tools are developed by using the Aspect Oriented programming platform. While creating the programs by the software developers, the shared memory resource may be accessed by multiple threads or tasks and cause data races (Basloom et al., 2023). The data race in a code indicates the presence of concurrency errors, and the data execution is affected by corrupting, hanging, and crashing the data (Al-Johany et al., 2023). It results in race condition, which is a multi-threaded bug that occurs when the order of the event is dominated by some undesirable tasks (Arteca et al., 2023), (De Sousa & Hasselbring, 2021). As data race is generally present in specific traces of threads, it becomes a challenging task for the developers to detect (Nithya & Chitra, 2020).

For detecting the data race, two different categories, such as static and dynamic detection approaches are usually followed. By using static detection, the variable name and source location of the accessed memory source can be analyzed. So, the turnaround time for the detection process is minimal (Paiva et al., 2020), (Bajaj & Sangwan, 2021). However, the static analysis model cannot perform well for larger programs and suffers from false positives (Moseler et al., 2022). The dynamic approach acts as a per-input / per-schedule detector, which detects the races by assessing all the possible thread schedules during program execution (Bai et al., 2022). This approach also has its own drawbacks as it analyzes races only for the selected input of code and takes a larger turnaround time. Hence, to enhance race detection further, hybrid tools and Machine learning techniques are recently explored (Almeida et al., 2021).

Using a Convolutional Neural Network (CNN), the raw data is efficiently learned, and the data race is detected at the code level and file level using a DeepRace model (Wang et al., 2022). The concurrently executing codes are analyzed through test cases, which are generated by using fuzzing approaches (Giebas & Wojszczyk, 2021). Further, the nested loops of code are parallelized using the frog leaping algorithm to detect the races (Zhang et al., 2021). The data race in multi-threaded code is also dynamically detected using the resettable encoded vector clock approaches (Pozzetti & Kshemkalyani, 2021), (Fava & Steffen, 2020). However, none of the existing works focused on handling the dynamic control flow of the AOP for examining the program paths and detecting the data races. Hence, in this work, a novel DRD framework is proposed using ECC-EVKMeans, SS-LSGRU, EVKM-BERT, and KL-FBIS techniques.

1.1 Problem statement:

The problems noticed in most of the existing works for data race detection are mentioned as follows;

[1] *Research Scholar, Jagannath University, Jaipur, Rajasthan
[2] Professor, Lovely Professional University, Punjab

❖ In (Ahishakiye et al., 2021), the dynamic control flow of the code was not effectively managed, thus resulting in higher false positives and false negatives.
❖ As data races were detected from the detection tools at the user level in (Mahjoub et al., 2022), the target of offload execution and the Open MultiProcessing (OpenMP) interfaces were not supported.
❖ Most of the works did not focus on identifying the harmful data races and benign data races.
❖ The dynamic detection techniques depend on limited input sets for the program execution, and so the program paths were not properly analyzed in (Jiang et al., 2022). So, undetected data races may result in inaccurate race identification with higher false negatives.

To overcome these limitations, the objectives concerned by the proposed framework for DRD are listed below;

❖ To reduce the false positives and false negatives, the control flow of the AOP is examined under the thread analysis to attain the context-sensitive code information using the Control flow graph.
❖ For supporting the target of offload execution and OpenMP tasks, the AOP dataset is trained by the proposed system and tested with the real-time AOP.
❖ The presence of all data races is accurately identified from the detected race condition using the proposed SS-LSGRU model.
❖ The thread analysis is carried out for AOP, and the optimal test cases are selected using the proposed G-CSOA for analyzing the program paths, which aids in accurate race detection.

The rest of the paper is aligned as follows: Section 2 discusses the related works of DRD; Section 3 elaborates on the processes of the proposed methodology and then the performance is assessed in section 4. Finally, the paper is wrapped with a conclusion and future scope in section 5.

2. LITERATURE SURVEY

(Mahjoub et al., 2022) detected the data races in concurrent programs by using the ConRacer prototype. The call graph of the Java program was constructed using the ConRacer tool by analyzing the control flow. Then, the escaped objects among threads were identified through the escape analysis. Further, the False Positives (FP) and False Negatives (FN) were minimized with the help of happens-before analyses. The performance was improved with lower FP, FN, and detection time. However, the test cases were not determined, so the detection of appropriate race conditions was not achieved.

(Jiang et al., 2022) presented a system for detecting concurrency errors in Multithreaded applications. Initially, a source code was acquired from a programming language. Then, the errors, such as insufficient mutexes, atomicity violations, and order violations of parallel threads were analyzed by using the rdao detector application tool. The system detected errors with better FP. Yet, the control flow of the code was not analyzed, which hinders the context-sensitive information for error detection.

(Tehranijamsaz et al., 2021) suggested an automated framework for detecting the race condition of the Interrupt-Driven Embedded Software. The input data was generated through the symbolic execution and the potential races. Then, the interrupts at potential racing points were validated and repaired using the virtual platform of SDRacer (Static and Dynamic Race detector). The performance was enhanced with enhanced accuracy and the least execution time. However, the FP and FN performance was not sufficiently achieved, which degrades the efficient detection process.

(Shi et al., 2021) explored a hybrid system called Hambug for detecting race conditions. The error in the source code was debugged using the Hambug tool, which provided the list of variables and the respective memory. The thread that occurred among the shared memory was tracked by the 'change detector'. Further, the race conditions were detected using the Shared Variable- Track algorithm. The system identified race conditions with higher efficiency. But, the optimal test cases were not identified, resulting in inaccurate detection of race conditions.

(Jin et al., 2023) recommended a hybrid-static analysis for identifying the data races. The driver code was statically analyzed at compile time to identify the variables. Then, the concurrent driver functions were identified through the entry and exit points of the driver function. Further, the data races were detected by analyzing the static lockset of the driver code. The performance was improved with better throughput. Yet, the model missed some driver codes and race conditions, thus limiting the detection of data races.

(Ahishakiye et al., 2021) presented an approach for analyzing the threads among concurrent Java programs. The test case was generated from a program. Then, the stack traces of the thread were analyzed, and the source code was extracted. Further, the thread information was accessed using the ThreadRadar method for debugging the multiple threads running in the programs. The approach identified threads with higher accuracy and less time. However, the number of variables and methods used in the program were not focused, which restricts the recognition of race conditions.

(Bora et al., 2021) suggested the test case selection for prioritization using the Discrete Cuckoo Search (DCS) algorithm. After selecting the test case, the test suite was prioritized. The problem that occurs with the ordering of test cases was minimized by transforming the real numbers into permutation sequences using the DCS

algorithm. The appropriate test case was selected using the algorithm with improved fitness. But, the DCS had a premature convergence problem, which degraded the selection of optimal test cases.

(Sulzmann & Stadtmüller, 2020) presented a web testing approach for generating the test case. The initial test suite was gathered from a Java file, and the abstract test cases were extracted. Then, the concrete test cases were obtained using the random data generators. Further, the candidate test cases were generated by applying the mutation operators. The performance was enhanced with better efficiency. However, the mutation function was sensitive to FP and consumed more time for developing test cases.

(Hao & Lu, 2021) selected the semi-automated test cases using the gradient descent approach. The test case was selected from the code by assuming the task as a multi-objective problem. Then, the optimal test cases were identified using the modified simulated annealing approach. The approach selected a test case with enhanced fitness and accuracy. Yet, the approach converged with the local optimum solution, which limits the selection of appropriate test cases.

(Yousaf et al., 2021) suggested a data retrieval model from the Java programming tool using BERT. The source code was accessed, and the locally cloned projects were extracted. Then, the vector values of the words were analyzed using a BERT model. The performance was improved with better throughput and word similarity. However, the multiple threads in the program were not detected; therefore, the required data was not accurately retrieved.

3. PROPOSED METHODOLOGY

In this work, the efficient DRD system is proposed by analyzing the variables, methods, threads, and test cases of AOP by using the SS-LSGRU, G-CSOA, and ECC-EVKMeans techniques. The proposed framework is represented as a block diagram in Figure 1.

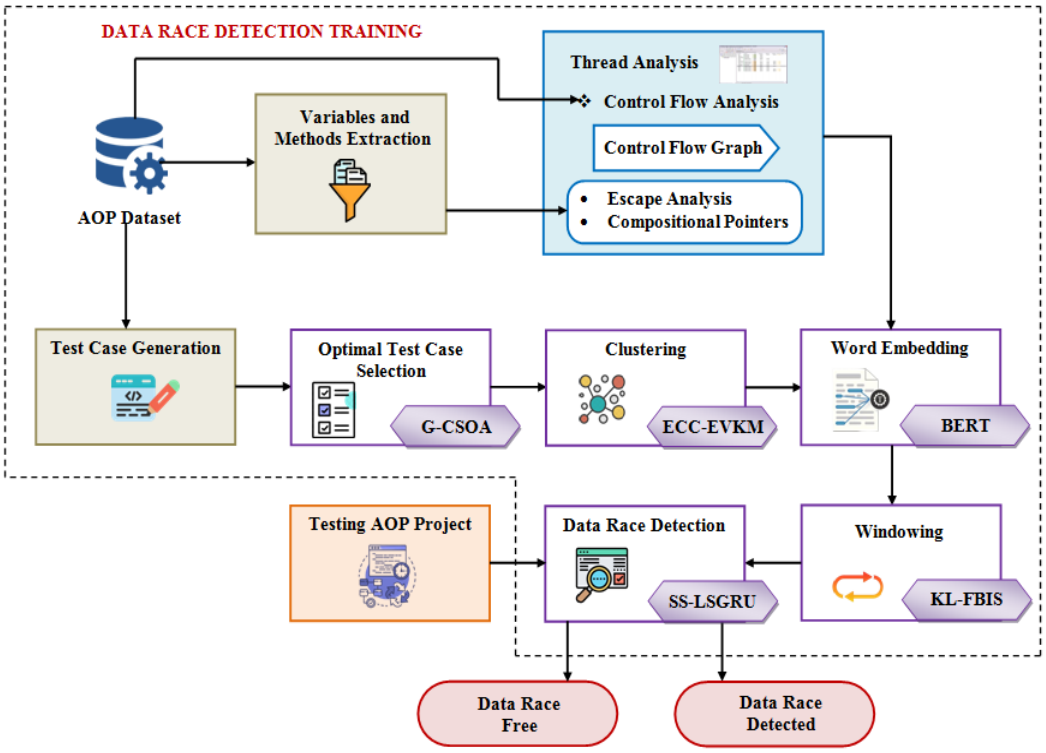

Figure 1: Structure of proposed workflow

3.1 Input data

Initially, an AOP compiled for different applications is considered as input data (J) for detecting the data races. It is expressed as,

$$J = \{J_1, J_2, J_3, \ldots\ldots, J_n\} \quad (1)$$

Where, n denotes the number of functions present in J.

3.2 Variables & methods extraction

From J, the number of variables, comprising the string values for executing the specific function of AOP and the methods, such as registration, Booking appointment, amount withdrawal, Balance checking, and so on of various applications compiled in AOP are extracted for further analysis. It is defined as,

$$\kappa = \{\upsilon_\ell, \omega\} \quad (2)$$

Where, κ indicates the extracted information about variables and methods from J, υ_ℓ denotes the ℓ number of variables, and ω specifies the methods in J.

3.3 Thread Analysis

Next, κ is fed as input to the thread analysis, which examines the execution of multiple threads running behind the source code J and the accumulation of memory. It is mentioned as,

$$\varsigma \in \kappa \quad (3)$$

Where, ς denotes the threads compiling behind the code.

Control flow analysis

Here, κ is given as input to the control flow analysis for analyzing the threads. The path of AOP, where different operations are executed, is analyzed using a Control Flow Graph (CFG) model. In CFG, every node illustrates the main function used in J; and the directed edges depict the jumps or the functions that alter the order of program execution by calling a function out of the loops in AOP. Further, the entry node and exit node of CFG represent the beginning and end of the program flow, respectively. The CFG model (ξ) is thus expressed as,

$$\xi = \{A_g, b_g, j_g, Z_g\} \quad (4)$$

Where, A_g denotes the entry node of the graph, b_g indicates the base function of code, j_g implies the directed edges of the graph, and Z_g specifies the exit node of CFG. To evaluate the control flow, any two edges of the graph should satisfy the condition (G_g). It is defined by,

$$G_g = \begin{cases} o_d(N_1) > 1 \\ i_d(N_2) > 1 \end{cases} \quad (5)$$

Where N_1, N_2 indicate any two nodes of the graph and o_d and i_d denote the outer degree and inner degree of nodes, respectively. Based on G_g, the control flow is analyzed, and it is mentioned as ψ.

Where, δ_e implies the escape states, $(\omega, \varsigma)_o$ denotes outside of the methods and threads, and $\arg$ indicates the argument. As the objective task or function of the program is retrieved through the cpt, a single parameterized analysis result for invoking the method is obtained and the in-build threads are suppressed. Thus, the analyzed parameter from cpt is declared as $\vec{p}_\omega$. Hence, the threads are analyzed from AOP, and it is mentioned as $\aleph$.

3.4 Test case generation

Next, the test cases are generated from J for accurately detecting the race conditions. Here, the test suites, including various test cases are developed for the earlier recognition of bugs in the code. It is specified as,

$$\Gamma_C = \{\Gamma_1, \Gamma_2, \Gamma_3, \ldots\ldots\ldots, \Gamma_q\} \quad (6)$$

Where, Γ_C indicates the generated test cases for various applications and q denotes the number of test cases.

3.5 Test case selection

From Γ_C, significant test cases are selected to enhance the detection of race conditions. For selecting the optimal test cases, a meta-heuristic Chameleon Swarm Optimization Algorithm (CSOA) is used owing to its tendency to solve global optimization problems. However, the CSOA used a fixed scaling factor during the eye rotation behavior, thus causing an incorrect rotation matrix and premature convergence. To overcome the issue, the Galois technique is used in the computation of the rotation matrix and attains the optimal solution.

Pseudo code of G-CSOA

Input: Generated test cases, Γ_C
Output: Optimal test cases, Γ_o
Begin
Initialize Test cases, $\Gamma_{initial}$
Initialize chameleon position, α_i
Evaluate fitness using eqn.(11)
$Fit = \max(\aleph)$
While $(itr < M_{itr})$
Determine hunting behaviour
If $Fit \neq \max(\aleph)$
Increase iteration
Update position, C_{new}^{itr+1}
Else
Original position
For $\hbar_{prey}$
Define Φ_{Gal} using eqn. (13)
$$\Phi_{Gal} = \frac{\sum_{i=1}^{\gamma}\left(\sqrt[5]{\Re_{ii}}\right)}{5}$$
Formulate $\Re_M$ along x and y axis
End for
For T_{BEST}
Evaluate tongue velocity, $V_{new}^{a,b}$
Calculate W
Determine γ
End for
$(itr) \rightarrow M_{itr}$
End if
End while
Return Γ_o
End

After that, the selected Γ_o are clustered according to similar applications to enable the proper data training and accurate identification of the data races. The clustering process is further described.

3.6 Clustering

Here, Γ_o is given as input to the clustering process. For clustering Γ_o, the k-Means algorithm is used, which efficiently handles the larger dataset with more dimensions. However, the k-Means algorithm is sensitive to outliers and does not perform well on uneven data. Hence, the variance problem of k-Means is resolved by utilizing the Entropy Variance (EV) technique. Further, the clustering accuracy of k-Means is improved using the Eisen Cosine Correlation (ECC) distance measure, which effectively measures the linear relationship between the test cases.

❋ After measuring E_{dis}, the problem of variance in Γ_o is tackled by introducing EV (E_{Var}), which provides the average distance among the test cases. Where, X expresses the probability of closer distance between Γ_o and C_{app} and $\log$ denotes the logarithmic function.

❋ Subsequently, a new centroid (C_{new}) is chosen among the clusters whenever the test cases are not grouped with the relevant application (C_{app}). Where, C_r indicates the centroid of each cluster and Γ_{total} implies the total number of Γ_o.

❋ The processes are repeated from the distance evaluation step until all the test cases are clustered with the appropriate application cluster.

3.7 Word Embedding

Next, the analyzed threads in AOP $(\aleph)$ and the clustered test cases (T_Γ) are represented as $(We)_{inp}$ and are fed as input to the word embedding phase to return the vector values of corresponding words. Here, the BERT

algorithm is used for the word embedding process, which processes the language and makes the data understandable by the DL network. The process of retrieving values using BERT is explained below;

Step 1: At first, the text format of $(We)_{inp}$ is divided into smaller tokens by the tokenization process and is indicated as $\wp$. Further, $\wp$ is converted into vector format using the embedded matrix. It is defined as,

$$\vec{\wp} = \chi_m(\wp) \tag{7}$$

Here, $\vec{\wp}$ denotes the vector value of $\wp$ and χ_m indicates the embedded matrix.

Step 2: Then, the correlation between the words of $\wp$ is estimated by calculating the self-attention score through the trainable parameters (Q,k) of BERT. It is expressed as,

$$S_{score} = \lambda\left(\frac{Qk^T}{\sqrt[2]{M_{\dim}}}\right) \tag{8}$$

Where, S_{score} indicates the self-attention score of each $\wp$, λ implies the softmax function, Q denotes the query, k^T mentions the transpose of the words' key, and $M_{\dim}$ specifies the dimension matrix of the key.

Step 3: Further, using λ, the tokens are activated along with the Euler's value. It is represented as,

$$\lambda(S_{score}) = \frac{e^{k_a}}{\sum_{b=1}^{t}\left(e^{k_b}\right)} \tag{9}$$

Where, e expresses the Euler's value, t denotes the total number of characters in $\vec{\wp}$, a denotes the a th character in $\vec{\wp}$, and b depicts the iterating index over the characters in $\vec{\wp}$.

Step 4: Then, the weights are applied over the BERT layers and the resultant outcome (R_O) is linearly transformed by residual connectivity $(\mathrm{Re}\, s)$ between the layers. It is represented as,

$$R_O = L_\tau(\mathrm{Re}\, s + S_{score}) \tag{10}$$

Here, L_τ denotes the linear transformation of the S_{score} of tokens.

Step 5: Among R_O, some tokens are masked to determine the missing attributes of the nearby words. So, the model is trained to estimate such masked tokens based on the embedding process. It is evaluated by,

$$Pb(\vartheta) = \lambda[L_\tau(R_o.S_{score})] \tag{11}$$

Where, Pb indicates prediction probability, and ϑ denotes the masked tokens.

Step 6: Lastly, the predicted ϑ are tuned based on pre-trained (Q,k) to exactly return the vector values. It is expressed as,

$$\vartheta_{tune} = [(Q,k) \approx (\vartheta)] \tag{12}$$

Whereas, ϑ_{tune} implies the fine-tuned outcome of the BERT model.

Hence, the vector values of the respective words in $(We)_{inp}$ are returned using the BERT algorithm and are represented as Ξ.

3.8 Windowing

Then, Ξ is given as input to the windowing process in which the nested loops are replaced with single loops to minimize the computational time by separating the vector values. For minimizing the nested loops, the windowing process is performed using the Fuzzy Interference System (FIS), which processes the uncertain information and makes relevant decisions. But, the FIS generates more non-feasible rules, which complicates the decision-making process. To overcome the issue, the KullbackLeibler (KL) Divergence and Bayesian techniques are used to quantify the deviations or correlations among the rules. The proposed KL-FBIS is further explained below;

3.9 Data Race Detection

Here, the β is given as input to the DRD phase for identifying the race conditions present in AOP. For recognizing data races, GRU is used, which can rapidly train the larger sequential data efficiently with its fewer parameters. However, GRU suffers from enormous data loss and gradient vanishing problems. To solve this issue, a Linear Scaling-based SoftSwish (LS-SS) activation function is introduced, and this proposed

detection framework is simply named as SS-LSGRU model. The architecture of the proposed SS-LSGRU classifier is represented in Figure 2.

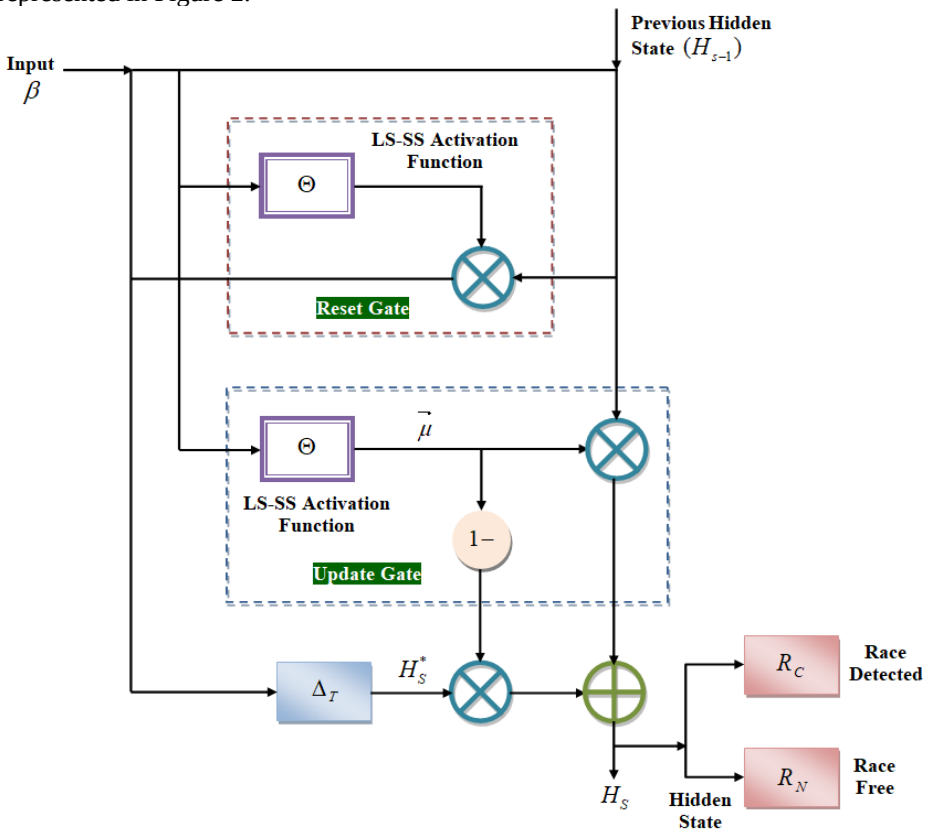

Figure 2: Architecture of SS-LSGRU

The process of the proposed SS-LSGRU classifier for DRD is described as follows;

Input layer

Initially, the β consisting vector values of threads and test cases in AOP are forwarded to the input layer to identify the data races. Next, the β is moved to the reset gate (χ_g), where it updates the memory regarding what information is to be retained or discarded from the network. Hence, the network is tailored with the different vector values for effectively learning the data. Here, Θ denotes the LSSS activation function, w_R indicates the weight of χ_g , and H_{s-1} implies the hidden state at the previous time step $(s-1)$. Here, the LS-SS activation is introduced instead of the sigmoid activation function to improve the data learning and solve the gradient vanishing problem. Where, y denotes the number of classes in β , Z_i , Z_j specify the similar attributes in β , and i and j indicate the number of vector values present in input and output layers, respectively.

Update gate

Further, the input data is passed through the update gate (Ω_g), which incorporates the candidate vector $(\vec{\mu})$ within $(0,1)$ into the hidden state (H_s) to determine how far the data is acquired for updating the hidden state.

Hidden state

Then, the hidden state is updated by $\vec{\mu}$ at every time step through the scaling of the hyperbolic tangent function (Δ_T) with the input data and the previous hidden state.

Output layer

Finally, the output layer determines what information is to be shared from the hidden state to produce the output for detecting the race condition (D_{Race}). (13)

Hence, the race condition is detected using the proposed SS-LSGRU network, and the presence of race condition is mentioned as R_C , and the absence of race condition is specified as R_N . For testing the AOP in real time, any developed AOP is fed into the proposed system, and the data race is identified based on the

trained SS-LSGRU classifier. Therefore, an efficient DRD system is formulated to prevent uncertain bugs and resource consumption in the AOP using the proposed work.

4. RESULTS AND DISCUSSIONS

The performance of the proposed DRD system is analyzed regarding various metrics in this section by comparing it with the existing techniques. For analysis, the proposed model is implemented using the PYTHON software tool, which integrates the system efficiently.

4.1 Dataset description

For performance analysis, a dataset named "Low-Level virtual machine Open MP Verifier (LLOV)" repository is utilized. This dataset is gathered from publicly available sources. It verifies the Open MP programs by using the polyhedral compilation technique. From the dataset, 80% and 20% of data are used for training and testing the proposed framework, respectively. The dataset link used for the proposed model is cited below in the reference section.

4.2 Performance assessment

Primarily, the performance of the proposed SS-LSGRU for the DRD is examined by comparing it with existing techniques, namely GRU, Bi-directional Long Short Term Memory (Bi-LSTM), Long Short Term Memory (LSTM), and Recurrent Neural Network (RNN).

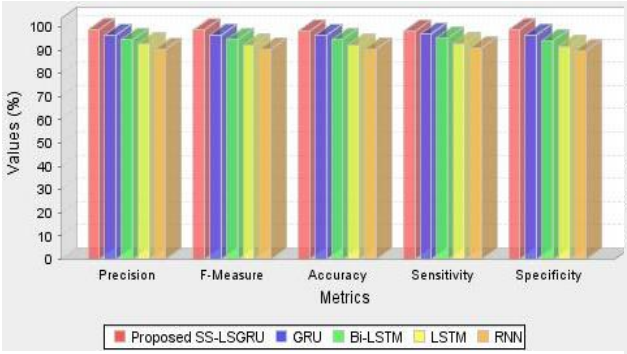

Figure 3 Analysis of proposed SS-LSGRU

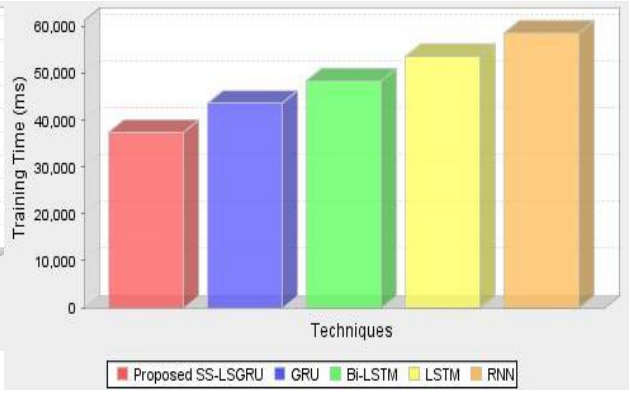

Figure 4: Training Time Evaluation

The proposed SS-LSGRU is analyzed regarding the accuracy, precision, f-measure, sensitivity, and specificity and is shown in Figure 3. The highest accuracy of 98.26%, the precision of 98.95%, the f-measure of 98.48%, the specificity of 98.78%, and the sensitivity of 98.02% are achieved by the proposed method. Meanwhile, the existing LSTM attained 93.03% accuracy, GRU attained 96.26% precision, and RNN attained 90.89% sensitivity. As the gradient vanishing problem and the data losses are resolved by the LS-SS activation function, the proposed algorithm detected race conditions with improved performance. Moreover, the lesser training time of 37751ms is taken by the SS-LSGRU model as represented in Figure 4 because of the selection of optimal test cases before training. The existing networks achieved an average training time of 50966 ms, which is higher than the proposed network. Thus, the proposed SS-LSGRU performs better than the prevailing models.

Table 1: Performance comparison of SS-LSGRU

Methods	Recall (%)	TNR (%)	PPV (%)	NPV (%)
SS-LSGRU	98.0273	98.7846	98.9565	98.3206
GRU	96.7132	96.6523	96.2648	96.2106
Bi-LSTM	95.1525	94.2158	94.3657	95.8216
LSTM	93.0378	91.6347	92.6478	92.8012
RNN	90.8957	89.6514	90.1054	89.3364

Table 1 depicts the performance of SS-LSGRU in terms of recall, True Negative Rate (TNR), Positive Predictive Value (PPV), and Negative Predictive Value (NPV). Also, 98.02% recall, 98.78% TNR, 98.95% PPV, and 98.32% NPV are attained by the proposed classifier. But, the existing methods, such as GRU attained 96.71% recall, Bi-LSTM attained 94.21% TNR, LSTM attained 92.64% PPV, and RNN attained 89.33% NPV, which are lower than the proposed technique. As every path of the AOP is determined through the CFG under control flow analysis, the proposed method detects race conditions with improved performance than the existing methods.

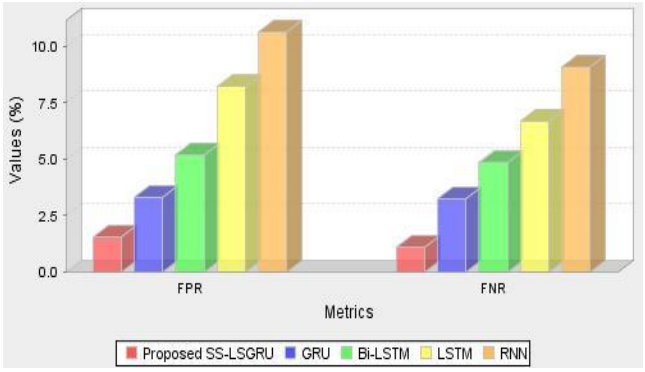

Figure 5: Graphical evaluation regarding FPR, FNR

The FPR and FNR achieved using SS-LSGRU are represented in Figure 5. The proposed network attained a lower FPR of 1.547 and FNR of 1.112. In the meantime, the existing GRU attained 3.326 FPR and LSTM attained 6.707 FNR, which are higher than the proposed network. As the test cases are grouped based on the applications and the vector values of words are returned prior to DRD, the proposed network detects data races with minimum error than the traditional networks.

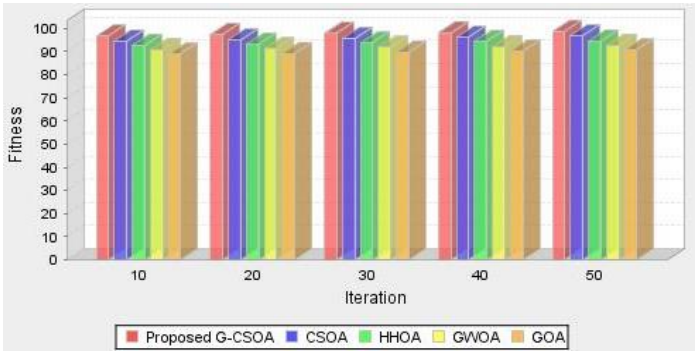

Figure 6: Fitness analysis of G-CSOA

Figure 6 illustrates the fitness performance of the proposed clustering algorithm for various iterations by comparing it with existing algorithms, such as CSOA, Harris Hawks Optimization Algorithm (HHOA), Grey Wolf Optimization Algorithm (GWOA), and Grasshopper Optimization Algorithm (GOA). The proposed algorithm achieved 98.87% fitness for 50 iterations because of improving the eye rotation scaling of CSOA using the Galois method, obtaining the optimal solution without premature convergence. Meanwhile, for 50 iterations, the HHOA attained 94.73% fitness and GOA attained 90.76% fitness. Also, the existing GWOA attained 91.78% fitness for 30 iterations. Hence, it is realized that the proposed algorithm achieved higher fitness for selecting vital test cases over the existing algorithms.

Table 2: Evaluation of proposed G-CSOA

No. of Iterations	**Selection time (ms)**				
	Proposed G-CSOA	**CSOA**	**HHOA**	**GWOA**	**GOA**
10	6325	9554	12547	15628	18327
20	10547	13659	16324	19487	22591
30	14269	17485	20957	23659	26487
40	18742	21659	24187	27845	30652
50	22359	25418	28653	31457	34187

The performance of the proposed G-CSOA is assessed regarding the selection time of test cases in different iterations and is exhibited in Table 2. It is observed that the optimal test cases are selected by the proposed G-CSOA within 6325ms in 10 iterations and 14269ms in 30 iterations. The existing algorithms, namely CSOA take 9554ms and GWOA takes 15628ms for 10 iterations. Further, the existing HHOA and GOA take 20957ms and 26487ms for 30 iterations, respectively. As the exploitation capability of the proposed algorithm is improved through the eye rotation behavior, the G-CSOA selected important test cases within a lesser duration than the prevalent algorithms.

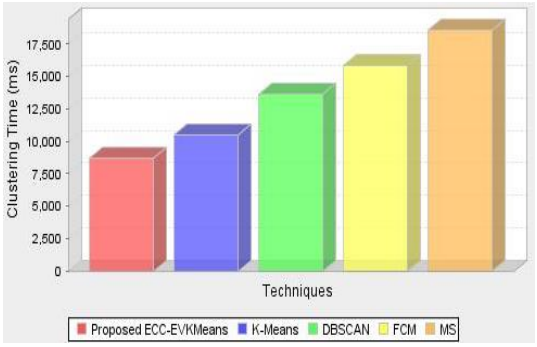

Figure 7: Clustering time analysis

The performance of the proposed ECC-EVKMeans algorithm for clustering the test cases with the appropriate applications is analyzed regarding clustering time and is shown in Figure 7. The clustering time of ECC-EVKMeans is determined by weighing against existing methods, namely K-Means, Density-Based Spatial Clustering of Applications with Noise (DBSCAN), Fuzzy C Means (FCM), and Mean Shift (MS). It is seen from Figure 7 that the proposed method achieved 8696 ms for clustering. But, the existing methods, such as K-Means took 10511ms, DBSCAN took 13616ms, FCM took 15838ms and MS took 18558ms, which are higher than the proposed method. Since the test cases in different units are evaluated by the ECC technique, the proposed algorithm achieved better performance over the conventional methods.

4.3 Comparative analysis

Here, the performance of the proposed DRD model is compared with the related works to verify the better performance of the proposed technique.

Table 3: Comparative analysis with related works

References	Technique	Precision (%)	Recall (%)
Proposed	SS-LSGRU	98.95	98.02
(Kumar et al., 2022)	MPI-ML	91.60	90.90
(Khanna et al., 2021)	TML	94	93
(Hirsch & Hofer, 2022)	OML classifiers	91	93
(Althiban et al., 2024)	PHT	-	-

Table 3 exhibits the performance comparison of proposed and existing DRD techniques regarding precision and recall. The performance is analogized with existing techniques, such as Event-based Statistical Analysis (E-SA), Message Passing Interface-based ML (MPI-ML), Traditional ML (TML) algorithms, Optimized ML classifiers (OML), and Parallel Hybrid Testing (PHT). It is noticed from Table 3 that the existing E-SA attained 90.9% precision, MPI-ML attained 91.6% precision, TML attained 93% recall, and OML attained 91% precision, which are lower than the proposed method. These existing techniques did not focus on different variables and paths of the program for DRD. As the proposed model analyzed every character in AOP along with its path through the control flow analysis, the precision and recall improved to 98.95% and 98.02%, respectively. Thus, the proposed model efficiently detects the race conditions than the existing techniques.

5. CONCLUSION

This paper developed an effective DRD system for the AOP by analyzing the control flow and optimal test cases using the proposed techniques. The introduced detection framework extracted the variables and methods and then analyzed every function, including the control flow of the AOP through the thread analysis. Furthermore, the test cases for different applications were generated from AOP. Optimal test cases were selected in the mean time of 14448 ms using G-CSOA with an average fitness of 97.85% for various iterations. Then, the test cases were clustered with relevant applications using the proposed ECC-EVKMeans within 8696ms. Further, the vector values of respective words in test cases and threads were retrieved using the BERT algorithm. Subsequently, the replacement of single loops over the nested loops by KL-FBIS. Thus, the race conditions were detected in 37751ms using the proposed SS-LSGRU network. Also, the data races were recognized using SS-LSGRU with an accuracy of 98.26%, precision of 98.95%, and recall of 98.02% along with 1.112 of FNR when analogized over existing techniques. Hence, for the AOP, the enhanced DRD is provided by using the proposed methodology.

REFERENCES

[1] Ahishakiye, F., Jarabo, J. I. R., Pun, V. K. I., & Stolz, V. (2021). Hardware-Assisted Online Data Race Detection.Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 108–126. https://doi.org/10.1007/978-3-030-87348-6_6

[2] Al-Johany, N. A., Eassa, F. E., Sharaf, S. A., Noaman, A. Y., & Ahmed, A. (2023). Prediction and Correction of Software Defects in Message-Passing Interfaces Using a Static Analysis Tool and Machine Learning.IEEE Access, 11, 60668–60680. https://doi.org/10.1109/ACCESS.2023.3285598

[3] Almeida, D. D., Braganca, L., Torres, F. S., Ferreira, R., & Nacif, J. A. M. (2021). HAMBug: A Hybrid CPU-FPGA System to Detect Race Conditions. IEEE Transactions on Circuits and Systems II: Express Briefs, 68(9), 3158–3162. https://doi.org/10.1109/TCSII.2021.3093985

[4] Althiban, A. S., Alharbi, H. M., Al Khuzayem, L. A., & Eassa, F. E. (2024).Predicting Software Defects in Hybrid MPI and OpenMP Parallel Programs Using Machine Learning.Electronics (Switzerland), 13(1), 1–31. https://doi.org/10.3390/electronics13010182

[5] Arteca, E., Schafer, M., & Tip, F. (2023).Learning How to Listen: Automatically Finding Bug Patterns in Event-Driven JavaScript APIs.IEEE Transactions on Software Engineering, 49(1), 166–184. https://doi.org/10.1109/TSE.2022.3147975

[6] Bai, J. J., Chen, Q. L., Jiang, Z. M., Lawall, J., & Hu, S. M. (2022).Hybrid Static-Dynamic Analysis of Data Races Caused by Inconsistent Locking Discipline in Device Drivers.IEEE Transactions on Software Engineering, 48(12), 5120–5135. https://doi.org/10.1109/TSE.2021.3138735

[7] Bajaj, A., & Sangwan, O. P. (2021).Discrete cuckoo search algorithms for test case prioritization.Applied Soft Computing, 110, 1–18.https://doi.org/10.1016/j.asoc.2021.107584

[8] Basloom, H., Dahab, M., AL-Ghamdi, A. S., Eassa, F., Alghamdi, A. M., & Haridi, S. (2023). A Parallel Hybrid Testing Technique for Tri-Programming Model-Based Software Systems.Computers, Materials and Continua, 74(2), 4501–4530. https://doi.org/10.32604/cmc.2023.033928

[9] Bora, U., Vaishay, S., Joshi, S., & Upadrasta, R. (2021). OpenMP aware MHP Analysis for Improved Static Data-Race Detection. Proceedings of LLVM-HPC 2021: 7th Annual Workshop on the LLVM Compiler Infrastructure in HPC, Held in Conjunction with SC 2021: The International Conference for High Performance Computing, Networking, Storage and Analysis, 1–11. https://doi.org/10.1109/LLVMHPC54804.2021.00006

[10] De Sousa, N. T., & Hasselbring, W. (2021). JavaBERT: Training a Transformer-Based Model for the Java Programming Language. Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASEW 2021, 90–95. https://doi.org/10.1109/ASEW52652.2021.00028

[11] Fava, D. S., & Steffen, M. (2020). Ready, set, Go!: Data-race detection and the Go language. Science of Computer Programming, 195, 1–23. https://doi.org/10.1016/j.scico.2020.102473

[12] Giebas, D., & Wojszczyk, R. (2021).Detection of Concurrency Errors in Multithreaded Applications Based on Static Source Code Analysis.IEEE Access, 9, 61298–61323. https://doi.org/10.1109/ACCESS.2021.3073859

[13] Hao, Z. Y., & Lu, F. M. (2021). Reverse Unfolding of Petri Nets and its Application in Program Data Race Detection. Ruan Jian Xue Bao/Journal of Software, 32(6), 1612–1630. https://doi.org/10.13328/j.cnki.jos.006240

[14] Hirsch, T., & Hofer, B. (2022).Using textual bug reports to predict the fault category of software bugs.Array, 15, 1–12. https://doi.org/10.1016/j.array.2022.100189

[15] Jiang, Z. M., Bai, J. J., Lu, K., & Hu, S. M. (2022).Context-Sensitive and Directional Concurrency Fuzzing for Data-Race Detection.29th Annual Network and Distributed System Security Symposium, NDSS 2022, 1–18. https://doi.org/10.14722/ndss.2022.24296

[16] Jin, F., Yu, L., Cogumbreiro, T., Shirako, J., & Sarkar, V. (2023).Dynamic Determinacy Race Detection for Task-Parallel Programs with Promises.In Leibniz International Proceedings in Informatics, LIPIcs (Vol. 263, pp. 1–30). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany. https://doi.org/10.4230/LIPIcs.ECOOP.2023.13

[17] Khanna, D., Purandare, R., & Sharma, S. (2021). Synthesizing Multi-threaded Tests from Sequential Traces to Detect Communication Deadlocks.Proceedings - 2021 IEEE 14th International Conference on Software Testing, Verification and Validation, ICST 2021, 1–12.https://doi.org/10.1109/ICST49551.2021.00013

[18] Kumar, S., Agrawal, A., & Biswas, S. (2022). Efficient Data Race Detection of Async-Finish Programs Using Vector Clocks. In PMAM 2022 - Proceedings of the 13th International Workshop on Programming Models and Applications for Multicores and Manycores, Part of PPoPP 2022 (Vol. 1, Issue 1). Association for Computing Machinery.https://doi.org/10.1145/3528425.3529101

[19] Lowe, D., Dubey, M. K., & Galhotra, B. (2020). Data race detection techniques in java: A comparative study. International Journal of Scientific and Technology Research, 9(2), 452–456.

[20] Mahjoub, S., Golsorkhtabaramiri, M., Amiri, S. S. S., Hosseinzadeh, M., & Mosavi, A. (2022).A New Combination Method for Improving Parallelism in Two and Three Level Perfect Nested Loops.IEEE Access, 10, 74542–74554. https://doi.org/10.1109/ACCESS.2022.3190483

[21] Moseler, O., Kreber, L., & Diehl, S. (2022). The ThreadRadar visualization for debugging concurrent Java programs.Journal of Visualization, 25(6), 1267–1289. https://doi.org/10.1007/s12650-022-00843-w
[22] Nithya, T. M., & Chitra, S. (2020). Soft computing-based semi-automated test case selection using gradient-based techniques.Soft Computing, 24(17), 12981–12987. https://doi.org/10.1007/s00500-020-04719-9
[23] Paiva, A. C. R., Restivo, A., & Almeida, S. (2020). Test case generation based on mutations over user execution traces. Software Quality Journal, 28(3), 1173–1186. https://doi.org/10.1007/s11219-020-09503-4
[24] Pozzetti, T., & Kshemkalyani, A. D. (2021).Resettable Encoded Vector Clock for Causality Analysis with an Application to Dynamic Race Detection.IEEE Transactions on Parallel and Distributed Systems, 32(4), 772–785. https://doi.org/10.1109/TPDS.2020.3032293
[25] Shi, Y., Wang, A., Yan, Y., & Liao, C. (2021). RDS: A cloud-based metaservice for detecting data races in parallel programs. ACM International Conference Proceeding Series, 1–10. https://doi.org/10.1145/3468737.3494089
[26] Sulzmann, M., & Stadtmüller, K. (2020). Efficient, near complete, and often sound hybrid dynamic data race prediction. ACM International Conference Proceeding Series, 30–51. https://doi.org/10.1145/3426182.3426185
[27] Tehranijamsaz, A., Khaleel, M., Akbari, R., & Jannesari, A. (2021). DeepRace: A learning-based data race detector. Proceedings - 2021 IEEE 14th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2021, 226–233.https://doi.org/10.1109/ICSTW52544.2021.00046
[28] Wang, Y., Gao, F., Wang, L., Yu, T., Zhao, J., & Li, X. (2022). Automatic Detection, Validation, and Repair of Race Conditions in Interrupt-Driven Embedded Software.IEEE Transactions on Software Engineering, 48(1), 346–363. https://doi.org/10.1109/TSE.2020.2989171
[29] Yousaf, M., Sindhu, M. A., Arif, M. H., & Rehman, S. U. (2021). Efficient Identification of Race Condition Vulnerability in C code by Abstract Interpretation and Value Analysis. 2021 International Conference on Cyber Warfare and Security, ICCWS 2021 - Proceedings, 70–75. https://doi.org/10.1109/ICCWS53234.2021.9702954
[30] Zhang, Y., Liu, H., & Qiao, L. (2021).Context-Sensitive Data Race Detection for Concurrent Programs.IEEE Access, 9, 20861–20867. https://doi.org/10.1109/ACCESS.2021.3055831
[31] Dataset link: https://github.com/utpalbora/LLOV/tree/master/test