Received: October 18, 2023. Revised: December 8, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

An Improved Deep Learning Based Test Case Prioritization Using Deep Reinforcement Learning

Ramakrihnan Shankar[1]* **Devarajan Sridhar[1]**

[1]*Department of Computer Science, Sri Krishna Adithya College of Arts and Science, Coimbatore- 641 042, Tamil Nadu, India*
* Corresponding author's Email: shankar.ramakrishnanphd17@gmail.com

Abstract: Continuous integration (CI) testing is crucial in modern software engineering and test case prioritization (TCP) techniques improve regression testing (RT) by prioritizing test cases (TCs). Various model has been developed to improve TCs failure prediction and prioritization in CI environments. But, prioritizing the TCs on large test suites without loss of information is a major challenging task. To address this, deep reinforcement prioritizer (DeepRP) model is proposed to improve prioritization in TCP on large test suites. This model employs deep reinforcement learning (DRL) model to learn more test case features, such as changes in source code, version control and code coverage. Also, it enhances self-optimization and adaptive ability for TCP. DRL training employs a deep neural network (DNN) structure to approximate various RL functions like value operation, Q function, transformation system and reward function. An RL system called Q-Learning which determines the appropriate action for an agent based on their action-value role. The DeepRP model uses test case features as input data and the priority of the test case as output. The action includes categorising TCs based on given scores, updating evaluations, calculating reward, and storing the chosen score in a temporary vector among the operations. The reward is computed based on the difference among the specified and ideal rankings for improved TCP on large test suites. The actions include sorting TCs based on assigned scores, updating observations, computing a reward and preserving the selected score in a temporary vector. The reward is calculated based on the distance between the assigned and optimal ranks for better TCP on large test suites. Finally, experimental results show DeepRP significantly achieves RMSE values of 0.09, 0.11 and 0.10 on paint control, IOF/ROL and GSDTSR datasets which is lesser than existing models algorithms like Deepgini, Hansie, DeepOrder, LogTCP and RL-TCP models.

Keywords: Continuous integration, Test case prioritization, Deep reinforcement learning, Deep neural network, Q-learning.

1. Introduction

Software testing is crucial in the software development process, detecting errors and defects in a system to ensure it works according to its specifications [1]. Regression testing (RT) is an essential process in software testing to prevent new bugs or errors and assures that changes to the program do not create any new challenges [2].

Currently, software projects often use CI, which automates and frequently performs software develops including RT [3]. The performance of all test scenarios is challenging due to resource, time, and expense constraints, and immediate software upgrade release cycles lead to reduced time for regression testing [4]. RT is also a frequent activity, especially in large software requiring significant resources and maintenance costs [5]. Methods for RT include minimization, selection and prioritization. Minimization eliminates redundant TCs [6], selection selects the most essential TCs [7] and TCP methods re-order a test suite to identify the best order of TCs, enhancing objectives like early failure recognition [8].

TCP methods are widely used in software sectors to improve regression testing productivity and quality [9]. They enable concurrent parallelization of debugging and testing software tasks, reducing the overall cost of testing. The TCP protocol allows testing to continue indefinitely until all resources are available or all tasks are executed [10]. TCP methods are divided into code and model-based approaches. Code-based methods [11] rely on source code applications to determine test execution, while model-based approaches [12] select tests based on anticipated system behavior models. On the other hand, TCP performs comprehensive analysis at every test until resources are exhausted or all tasks are executed

Artificial intelligence (AI) techniques including machine learning (ML) and deep learning (DL), have been successfully used to reduce software engineering effort and lower software failure rates [13]. ML-based TCP techniques [14] automate various activities, relying on easy-to-compute features and practical data like history data implementation, full-ranged statistics, code complications and interpretative data. However, ML approaches may degrade performance for large-scale test suites with additional loss of information.

DL-based TCP approaches are used to forecast TCP in large test suites based on variables like test length and execution conditions [15]. One of the advanced DL model i.e., RL has shown significant interest in RT by constantly adjusting priority techniques [16]. RL models consistently and autonomously train the TCP approach, achieving beneficial ranking accuracy of regression TCs. However, RL-based algorithms provide efficient results in TCP for CI, but tuning and optimization of hyper parameters are challenging. RL results lack scalability and can't manage large-scale test suites, making the network structure complex with lower fault detection time and rate.

To resolve this issue, DeepRP is a proposed model that integrates RL and DNN-based TCP methods to improve the accuracy of prioritization in test case prioritization. It uses DNN-based TCP with RL to learn additional properties of TCs, such as changes in source code, version control and code coverage. This model also enhances the self-optimization and adaptability of TCP, reducing software failure risk. A DNN is used in DRL training to estimate all RL functions, such as the conversion mechanism, reward operation, value function, and Q function. A RL system called Q-Learning uses an action-value role to select the appropriate course of action for an agent. The model uses activities such as saving scores, updating observations, computing rewards and sorting TCs based on their assigned scores. This model offers solutions to enhance the accuracy and stability of deep learning models in CI testing for TCP for large-sized test suits in software TCs.

The rest of the paper is organized as follows: The investigations on TCP prediction using DL algorithms are presented in Section II. The proposed method is covered in Section III, and its performance in comparison to the existing algorithms is shown in Section IV. The study's conclusion and recommendations for improvements are provided in Section V.

2. Literature survey

A test prioritization method called DeepGini was devised [17] to prioritize the DNN tests based on statistical views for high-dimensional object classification. It reduces misclassification chances and measures set impurity, identifying likely-misclassified tests quickly. But, there was a lower ratio of faults\errors were identified by this model.

A scalable model for CI and RT in IoT-based applications was presented [18] based on IoT-related TCP and evaluation parameters. This model used search-based algorithms to determine optimal priority ordering for TCs, followed by a trained predictive model using DL models to ensure system efficacy. But, time consumption was increased due to the absence of computationally intensive tasks during prioritization.

The Hansie model was constructed [19] for prioritizing composite and consensus regression tests. This algorithm uses priority-aware hybridization and priority-blind computation for consensus sequence computation. It conducts integrative tests using normal and abnormal windows. But, decreased prioritization efficacy were resulted on fault recognition and detection time.

A test prioritizing approach called RLTCP was suggested [20] which reduces test failures while reducing tests. They created a weighted coverage graph to characterize the relationship between TCs for user interface evaluation. RLTCP merged RL with the graph, but not calibrated RL hyperparameters. But, only individualistic unit TCs and minimal number of reward functions were considered.

A DL-based regression framework called DeepOrder was developed [21] for prioritizing the regression tests in CI. The DNN was trained using historical test data, including time and completion status of TCs to identify failed cases and important ones within a specific test suite. Consequently, this model result with lower rate of fault identification and detects faults at slow process.

A new learn-to-rank approach was constructed [22] using the extended finite state machine (EFSM) for TCP. The random forest approach included heuristic prioritizing schemes, but it did not consider

Table 1. List of notations

Notations	Description
T	Test Suite
TCs	TCs
PT	Prioritizations Of T
t	Time
s_t	Action
s_t	State
s_{t+1}	Next State
$N!$	State Space Dimension
T^{pass}	Passing TCs
T^{fail}	Failing TCs
p_t	Total Degree Of Penalization
C	Executing Command
s^f	Weights Of Failing Test Step
a^f	Initialized Passing TCs
r	Reward Function
$t \in [0,1]$	Transition Operation ($t \in [0,1]$)
γ	Discount Factor
$V^{\pi}(s)$	Search Policy by Value Function
π^*	Optimum Policy
G_t	Agents Expected Return
Q^*	Optimal Q Function
Π	Policy
α	Learning Rate
θ	$Q -$ Network Parameter
AV	Action Vector
QV	$Q -$ Vector
$\bar{\theta}_k$	$Q -$ Function value at k^{th} Iteration
$\mathcal{B}$	Batch Size
Q_T	Target Q
$Loss$	Loss Function

the time cost of TC execution or fault security level, which could affect construct validation. However, this model results significant time complexity issues when working on large datasets.

Two TCP dynamic sliding window techniques like test suit and individual TC-based dynamic sliding window was introduced [23]. This model initially used a fixed-size sliding window for all CI tests but later developed adaptive approaches. The performance might influence on construct validation owing to its inability for test case execution time cost and fault security level.

A conceptual data model was suggested [24] for retrieving data sources and their connections in a standard CI environment. This model defined a set of characteristics used in related investigations, applied to train ML models and accurately prioritized TCs. More advanced techniques were needed to improve the efficiency of real-time TCP.

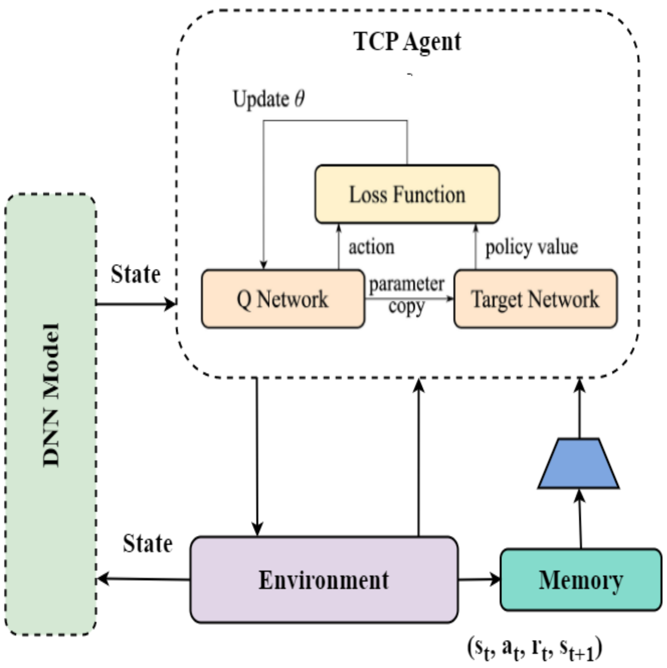

Figure. 1 Block structure of the DeepRP model

A novel black-box TCP (BTCP) model known as LogTCP was devised [25] comprising log pre-processing, log representation and TCP modules. The LogTCP model was utilized to implement various log-based BTCP schemes, combining various log representation methods and prioritizing approaches. However, this model results with lower average percentage of fault detected (APFD) values.

The TCP model using an optimization technique and an RL model was initiated [26] to handle large scale test suites. This model complied log files of developers and users using activity tracking technologies by using the RL model to determine future rewards and used an error seeding approach to check software specialist performance. But, this model enables lower fault detection rates while increasing the cost and time of prioritization mechanisms.

3. Proposed methodology

In this part, the complete structure of the DeepRP model is illustrated and depicted in Fig. 1. Table 1 outlined the notations utilized in this study.

3.1 TCP framework

The primary goal of TCP is to identify an optimal order of TCs series to enhance some performance indicator. TCP techniques in the CI sector are

Received: October 18, 2023. Revised: December 8, 2023.

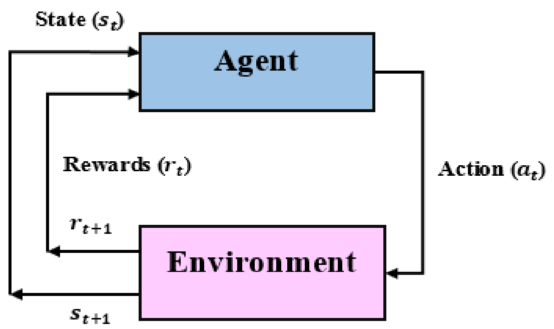

Figure. 2 Reinforcement learning model

Table 2. Two test case samples

Test case 1	Test case 2
<Input textbox username> (A)	<Click button Login> (E)
<Click button Login> (B)	<Click button Setting> (F)
<Input textbox Search> (C)	<Click button Change Password> (G)
<Click button Search> (D)	

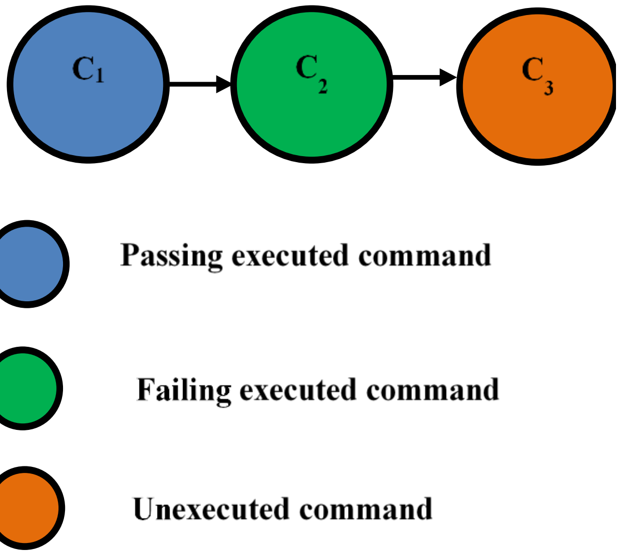

Figure. 3 TCs command

Table 3. Adjacency Matrix

	C_1	C_2	C_3
C_1	0	1	0
C_2	0	0	0
C_3	0	0	0

designed to detect problems as soon as they occur. For a test suite T, an array of each potential prioritizations (orderings) of T, (PT) and a function f that quantifies the efficiency of a particular prioritization from PT to a real number, which is obtained as follows,

$$T' \in PT \; s.t. (\forall T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')] \qquad (1)$$

The TCP issue aims to achieve the optimal T' for RT, but insufficient resources make it impossible to implement an entire suite. Time-consuming TCP methods can be beneficial as they maintain coverage and allow instant execution of failed TCs, reducing resources and costs while ensuring maximum fault coverage.

3.2 Reinforcement learning model

A RL model consists of four key components: the agent, who makes decisions and interacts with its environment. The agent takes an action a_t depending on the environment's state s_t at each time step t. It then moves to the subsequent state s_{t+1} and sends out a reward indication. The agent aims for an accumulated sum of rewards over an immediate reward, and its objective is to increase the aggregated rewards it obtains across different time intervals. The Fig. 2 depicts the RL model.

The elements of TCP challenges are listed below.

States: Each variation of the source TC is specified as a state which means that the dimension of the state space is N for a test suite T with TC is given as $TC\ (N!)$. This is because the improvisation unit of the problems is represented by each iteration as one TCs sequences. Two instances of tokenized instructions as characters are shown in Table 2. It should be noted that command B is allocated by two TCs.

Reward: The weighted transitions between the evaluation cycles are used by the designed system to evaluate the TCs. The full-ranged graph records the weights, and changes to the graph's values are the outcome of both rewards and penalties. As shown in Fig. 3, each matrix cell represents a single step on the path from successful to unsuccessful command execution. On this basis, the two reward functions given in Eq. (2) and Eq. (3) are carried out during the graph's initialization.

$$\forall_{s}, s^{f} \in t\big(\big(t \in T^{fail} \wedge s \leq s^{f}\big) \Rightarrow increase\_weight(s, a^{f}) \qquad (2)$$

$$\forall_{s} \in t\big(\big(t \in T^{fail} \wedge s \leq s^{f}\big) \Rightarrow increase\_weight(s, a^{f}) \qquad (3)$$

The term $s \leq s'$ determines the event s occurring before action s' in specified TCs. The incentive for submitting TCs is outlined in Eq. (2). The weights of Table 3 specifies the association matrix for the full-ranged graph after the TCs are processed shown in Fig 3.

failing test steps s^f are initially set to an expected amount a^f for each TC at t in the collection of failing TCs, T^{fail}. However, the test steps associated with successful TCs are augmented to a value a^p as shown in Eq. (3). By selecting $a^f > a^p$, the test steps with greater weights will be carried out with greater priority. Initial training includes an assessment of Eq. (2) and Eq. (3). To penalize the incorrectly arranged succeeding TCs, Eq. (3) is carried out in repeated cycles.

As demonstrated in Eq. (4), as given T as a test suite where T^{pass} and T^{fail} represents the portion of passing and failing TCs respectively. The system penalizes failed tests cases, TC^f with a constant p for each lower-rated case.

$$\forall_t \in T^{pass}, p_t = p * \sum_{t^f \in T^{fail} \wedge rank(t^f) < rank(t)} 1 \quad (4)$$

Where, the penalty severity is denoted by p_t. The rank function maps the order of TCs in the test suite. If fine-tuning is needed, a penalization variable p is generated. The sequence eights in succeeding TCs are reduced by p_t for each passing case penalized. The system prioritizes succeeding tests over failing ones.

Actions: An action is a signal for the RL system to change states, which are variations of the input test suite. This method permits unlimited state transitions as the prioritizing model which has the potential to arbitrarily reconstruct the input test suite.

Policy: The policy of this model examines the actual coverage graph for action preference. The system calculates the total weight of the steps in each specific TC by estimating the number of steps in that TC. In the final test suite, the TC stands out more when the value is higher. The system connects TCs with the similar value in arbitrary state. From the TCs in Table 3, the ranking value of the TCs is 5, since it includes two sequence $C_1 - C_2$ and $C_2 - C_3$ with weights of 2 and 3, respectively.

3.3 Deep reinforcement learning

RL models can train without feature construction, face challenges in high-dimensional data and dynamic environments. Misclassification is a problem with DL's ability to recognize complex patterns. New learning methodologies have emerged by integrating RL models with DNN, benefiting function approximation and RL-based DNN training. DRL algorithms offer efficient solutions compared to traditional methods in accessing the TCP on larger suites.

The DRL problem may be described more readily as a Markov Decision Process (MDP) employing the five-tuple $(s, a, t, r,)$ denoting the state space, action space, transition operation ($t \in [0,1]$), reward operation and discount variable ($\gamma \in [0,1]$). In order to maximize its expected return, an RL agent can use the value parameter $V^{\pi}(s)$ as shown in Eq. (5).

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \,\middle|\, s_t = s.\pi\right) \quad (5)$$

$$V^* = \max_{\pi \in \Pi} V^{\pi}(s) \quad (6)$$

$$r_t = a \sim \underset{\pi(s_t,.)}{\mathbb{E}} r(s_t, a, s_{t+1}) \quad (7)$$

$$\mathbb{P}(s_{t+1} | s_t, a_t) = T(s_t.a_t.s_{t+1}) * a_t \sim \pi(s_t. \quad .) \quad (8)$$

Thus, the operation of action-value $Q(s, a)$ is stated as follows,

$$Q^{\pi}(s.a) = \mathbb{E}_{\pi}\left(\left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \,\middle|\, s_t = s.a_t = a.\pi\right)\right) \quad (9)$$

$$Q^* = \max_{\pi \in \Pi} Q^{\pi}(s.a) \quad (10)$$

Policy iteration is an efficient method for addressing MDP, aiming to reach the superlative policy (π^*). Value iteration involves establishing random values and constantly iterating to compute an enhanced state or action-value function, providing the optimal policy and its value using the Bellman expectation equation which is stated as follows,

$$V^{\pi}(s) = \mathbb{E}_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s) \quad (11)$$

The optimal policy is improved by using a greedy action to increase the state-action value for policy testing tasks. Model-free techniques may be utilized in undefined scenarios where the action-value function, rather than the state-value operation which can be enhanced to identify the superlative policy (π^*) is defined in Eq. (12).

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}(r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, \ a_t = a) \quad (12)$$

The agent will construct a sequence of changes where the reward increases from $\{r_1, r_2, \ldots\ldots r_t, \ldots\ldots r_T\}$ for $t = 1, 2, \ldots. T$. In the meantime, the equation for the discounted value G_t, which stands for agent's projected return, is as follows:

$$G_t = r_t + \gamma . r_{t+1} + \gamma^2 . r_{t+2} + \cdots + \gamma^{T-t} . r_T \quad (13)$$

Where the parameter γ will be the discount factor ($\gamma \in [0,1]$).

3.4 Q-Learning

Q-Learning is used to optimize the TCP's action-selection policy, guiding the agent in decision-making based on guidelines. The reward r is used to evaluate efficiency, with the purpose of informing the agent to maximize cumulative reward over time. Q-learning applies its Q-values to the provide solutions of RL issues. The quality function (Q-function) also known as the action-value function necessitates to be specified for each policy P. $\Pi\ (s_t, a_t)$ represents the predicted accumulated reward that might be attained by performing a series of procedures starting with action a_t from s_t; and then following to the policy Π. The best Q function, denoted by $Q^*$is the one that maximizes the predicted progressive reward for a given state-action conjunction under all policy choices.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q^*(s,a) - Q^*(s,a)) \quad (14)$$

$$Q^*(s,\ a) = \max_{\pi} \sum_{t>0} (\gamma^t r_t \,|s = s_t,\ a = a_t, \pi|) \quad (15)$$

The optimum strategy Q^* is defined as actions taken at each time step Q^* that increase the sum of $r + Q^*\ (s_t + 1, a_t + 1,)$ where r represents the immediate reward, t and $t + 1$ represents the current and next time step respectively. The discount value (γ) is used to balance long-term benefits with short-term ones in a test suite environment, where the state represents RT operations and agents interpret TCs as agents, with actions representing the range of possible actions.

3.5 Deep Q-network (DQN)

DQN is a value-based DRL algorithm which integrates Q-learning with DNN and experience replay. It addresses instability using the approximation learning state-action value function technique. DQN randomly batches samples between N data points and training experience. Deep Q-learning (DQL) uses the Q-function in conjunction with DNN, with $Q(s, a, \theta)$ indicating the Q-learning agent. The DQL agent has a neural network θ that helps decide actions and rewards them, using a variable to represent the weights of each layer and the model's subsequent state s_{i+1} or s'. The DNN's target weights are regularly updated through a feedback loop to determine the desired Q-value, ensuring accurate estimation and forecasting. Consider the target function of DQL as in Eq. (16),

$$Y_k^Q = r + \gamma \max_{a' \in A} Q(s', a'\,;\, \overline{\theta}_k) \quad (16)$$

At the same time, the stability is maintained and the possibility of divergence is minimized by only updating the variable $\overline{\theta}_k$, which initiates the coefficients of the Q-function at the k^{th} iteration for each $A \in N$ iterations. DQN uses target Q-network and replay memory heuristics to minimize instabilities, while other heuristics like clipping rewards are employed to maintain practicality and ensure appropriate learning, as illustrated in Fig. 4.

The input layer of the DNN architecture receives the attributes (i.e., variable states) of the present state and the output layer predicts the Q-values using the DNN parameters and weights.

For every record i in the active state, the action with the greatest Q values is selected as

$$AV_i = argmax(QV_i), \qquad \forall_i \in \mathcal{B} \quad (17)$$

In above Eq. (17), AV and QV represents the action vector and Q-vector, $\mathcal{B}$ is the batch of TC data which will fed into the DNN layer for TCP. The reward mechanism is used to compute rewards by evaluating AV_i with the class in the dataset for relevant TCs and the AV will be generated either at arbitrary or using DQN predictions which is depicted in Fig.5. For the next state, the DQL agents necessitates to evaluate the training tasks for prioritizing the TCs on substantial test suites which is designated as $Q'V_i$ and $A'V_i$ for all $i \in \mathcal{B}$. Based on the rewards, discount level for potential rewards and expected Q-vectors for the target Q (Q_T) is given in Eq. (18),

$$Q_{T_i} = r.V_i + \gamma . Q'V_i \quad (18)$$

The output of Q_{T_i} is given into the DQN during the learning improvement phase of TCP to be utilised

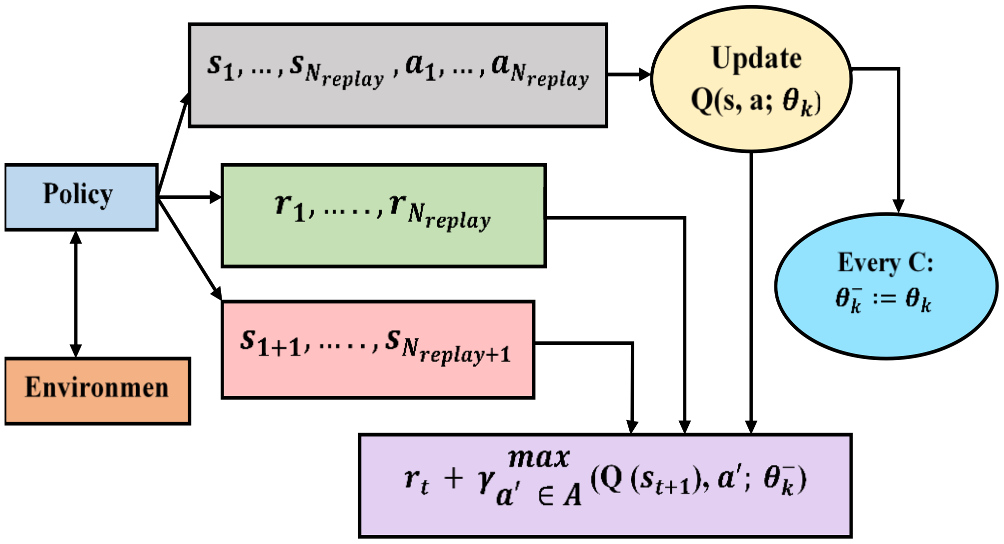
$s_1, \ldots, s_{N_{replay}}, a_1, \ldots, a_{N_{replay}}$
Update Q(s, a; θ_k)
Policy
$r_1, \ldots, r_{N_{replay}}$
Every C: $\theta_k^- := \theta_k$
$s_{1+1}, \ldots, s_{N_{replay+1}}$
Environmen
$r_t + \gamma \max_{a' \in A} (Q(s_{t+1}), a'; \theta_k^-)$

Figure. 4 DQN structure

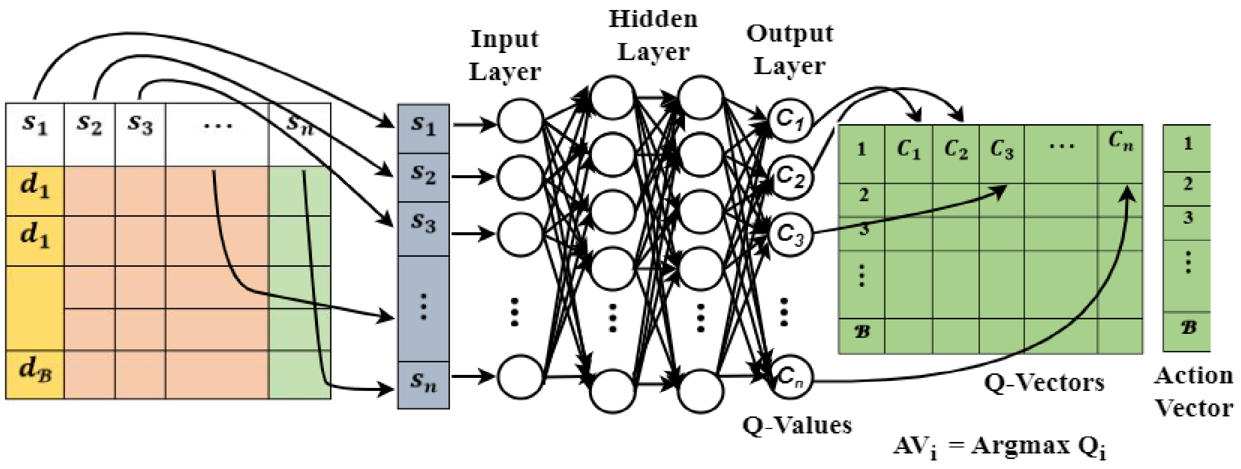

Figure. 5 DQN model detection employing states and DNN, the results are $Q -$ values, and actions are generated using the current state's argmax Q_i.

in the training phase and the calculation of the loss function.

Mean square error (MSE) loss is a technique in neural network training that minimizes the difference between predicted and final Q-values. It is calculated by adding the reward from the current state to the expected Q-value for the subsequent state and multiplying by γ. It is crucial for assessing DQN performance after each task iteration. After each task iteration in a DQN network, a loss value is calculated using the recent states and the desired network. Eq. (19) depicts the loss operation.

$$Loss = \frac{1}{n}\sum_n (Q(s,a) - r + \gamma Q(s',a'))^2 \quad (19)$$

The DeepRP is trained to rank the TCs using a Q-function to map each state to a single $Q -$ value representing all feasible transitions. The highest Q-value is used to decide the recommended course of action. Training the model involves iterations and episodes in order to encompass the complete dataset. This DeepRP model improves TCP for large-scale test suites by learning additional aspects of TCs, such as source code modifications, version control and code coverage, eliminating the drawbacks of RL and DNN for better TCP on large test suites.

4. Result and description

4.1 Dataset description

In order to demonstrate the efficacy of the proposed methodology, an industry data sets Paint Control, IOF/ROL [27] and the Google Shared Dataset of Test Suite Results (GSDTSR) [28] are utilized. The databases provide data from over 300 CI cycles, including past test executions and produced results. Table 4 displays the general layout

Table 4. Comprehensive Overview of Industrial Data Sets: Quantity of data is displayed in each column.

Dataset	TCs	CI	Verdicts	Failed
Paint Control	114	312	25,594	19.36%
IOF/ROL	2,086	320	30,319	28.43%
GSDTSR	5,555	336	1,260,617	0.25%

Table 5. Parameter settings for existing and proposed model

Models	Parameters	Range
Deepgini [17]	No. of convolutional layer	2
	No. of hidden layer	4
	Batch size	100
	No. of epoch	50
	Learning rate	0.001
	Loss Function	MSE
Hansie [19]	Test Agreement score	16
	Window size	(2,2,1)
	Learning Rare	0.0001
DeepOrder [21]	No. of. Hidden Layer	45
	Epochs	1000
	Loss function	MSE
	Optimizer	Adam
	Learning Rate	0.001
LogTCP [25]	Total states number	11
	Action number	3
	Agent Episodes	1000
	Test Sequences	40
	Learning rate	0.001
	Loss Function	SGD
RL-TCP [16]	Reward value	0.5
	Action value	4
	State number	9
	Discount factor (γ)	0.99
Proposed DeepRP	Episodes	700
	Time step in episode	0.5
	Total state number	50
	Action size	10
	Optimizer	ReLU
	Memory size	175
	Learning rate	0.001
	Batch size	64
	Loss Function	MSE

of the data sets, including all case scenarios featuring software.

4.2 Performance evaluation

The performance of DeepRP existing algorithms like Deepgini [17], Hansie [19], DeepOrder [21], LogTCP [25] and RL-TCP [16] is executed in Python 3.7.8 using the dataset mentioned in section 4.1. Table 5 lists parameter settings for the proposed DeepRP and existing models.

The proposed and existing models are evaluated by different metrics which is briefly given below.

4.2.1. Average percentage of faults detected (APFD)

It determines how rapidly a test suite finds errors. During a test suite execution, the weighted average of the identified error is computed using APFD. The formula for APFD is

$$APFD = 1 - \left(\frac{TF_1 + TF_2 + \cdots + TF_m}{NM} + \frac{1}{2n}\right) \tag{20}$$

In above Eq. (20), T is the resulted test suite; m will be the total number of errors detected from the program for TC execution. n be the total TC number, $Tf_1, Tf_2, \ldots., Tf_m$ are the points of initial test T which reveals the fault m.

4.2.2. Average percentage of faults detected per cost ($APFD_c$)

It accurately compares the typical percentage of TCs that cost money to the typical percentage of fault severity found. Eq. (21) yields the weighted (cost-conscious) average percentage of errors discovered throughout the execution of test suite T'.

$$APFD_c = \frac{\sum_{i=1}^{m}(f_i * (\sum_{j=TF_i}^{n} t_j - \frac{1}{2} tTF_i))}{\sum_{j=TF_i}^{n} t_j * \sum_{i=1}^{m}(f_i)} \tag{21}$$

In the preceding Eq. (21), T is the test suite including n TCs with costs $t_1, t_2, \ldots \ldots, t_n$. Assume F be the collection of m faults exposed in T and the severities of those faults be f_1, f_2, ….., f_m. The first TC that finds the problem i is . TF_i.

4.2.3. Time-aware average percent of faults detected ($APFD_{TA}$)

It is the particular case of $APFD_c$ for the instances with uniform test costs and fault severities. The notation of $APFD_{TA}$ is illustrated in Eq. (22),

$$APFD_{TA} = \frac{\sum_{i=1}^{m}(\sum_{j=TF_i}^{n} C_j - \frac{1}{2} CTF_i))}{\sum_{j=1}^{n} C_j * |\sigma|} \tag{22}$$

Where, σ is constant between the first and last TCs in the complete test suites.

4.2.4. Root mean square error (RMSE)

It the variation among the expected and identified NAPFD values. The different value calculated for T'

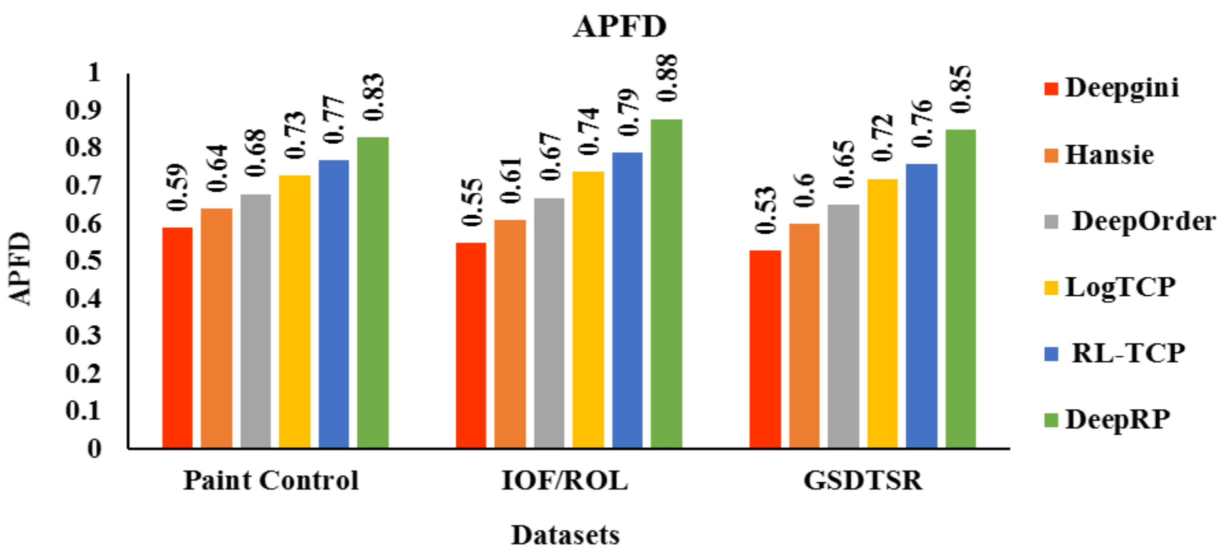

Figure. 6 Comparison of APFD Metric for given datasets

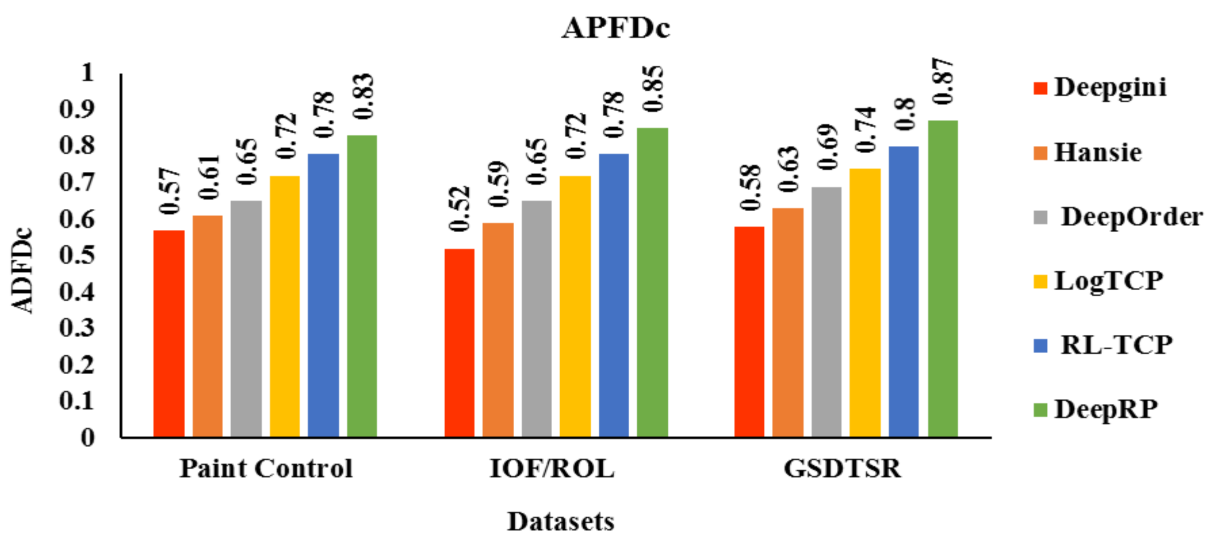

Figure. 7 Comparison of $APFD_c$ Metric for given datasets

in a CI iteration q obtained by learning model $(\hat{u}_C)$ and the closest expected value T' is determined by the RL model. The RMSE value is computed as follows,

$$RMSE\ (\Psi) = \sqrt{\frac{\sum_q^{CI}(\hat{u}_q - u_q)}{CI}} \tag{23}$$

In Eq. (23), CI is the number of CI cycles in a system. Lower RMSE values indicate more accurate algorithms. Because finding an ideal priority is a difficult undertaking, $\hat{u}_q$ determined via RL is an estimate of the optimal prioritization.

4.2.5. Time complexity

This model operates in n steps, where n is the number of TCs. Each step involves selecting the preceding TCs and updating the coverage of the existing TCs in $O(nm)$ time. As a result, the overall time complexity of this model is stated as is $O(n^2m)$.

The Fig. 6 demonstrates the comparison of APFD values of proposed and existing models on different TCs dataset. From this analsysis, is indicated that the proposed DeepRP models realizes higher effectiveness in prioritizing the TCs for large test suites effectively than other classical models. For instance, the APFD value of DeepRP is 40.68%, 29.69%, 22.06%, 13.69% and 7.79% higher than the Deepgini, Hansie, DeepOrder, LogTCP and RL-TCP models respectively on paint control dataset.

Similarly, the Fig. 7 demonstrates the comparison of $APFD_c$ values of proposed and existing models on different TC dataset. It is proved that the proposed model achieved greater $APFD_c$ value than other existing models. For instances, the $APFD_c$value of

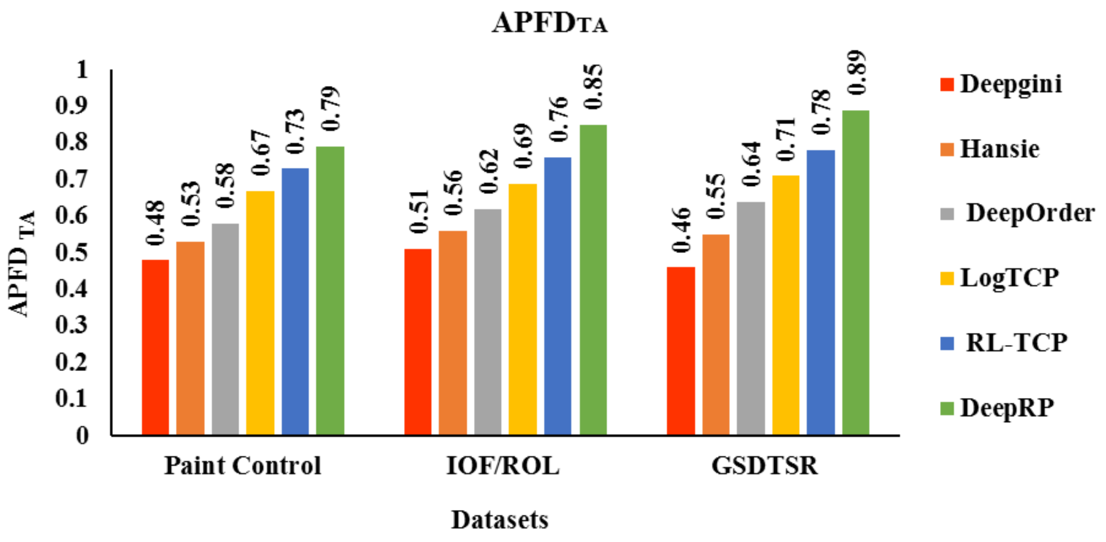

Figure. 8 Comparison of $APFD_{TA}$ Metric for given datasets

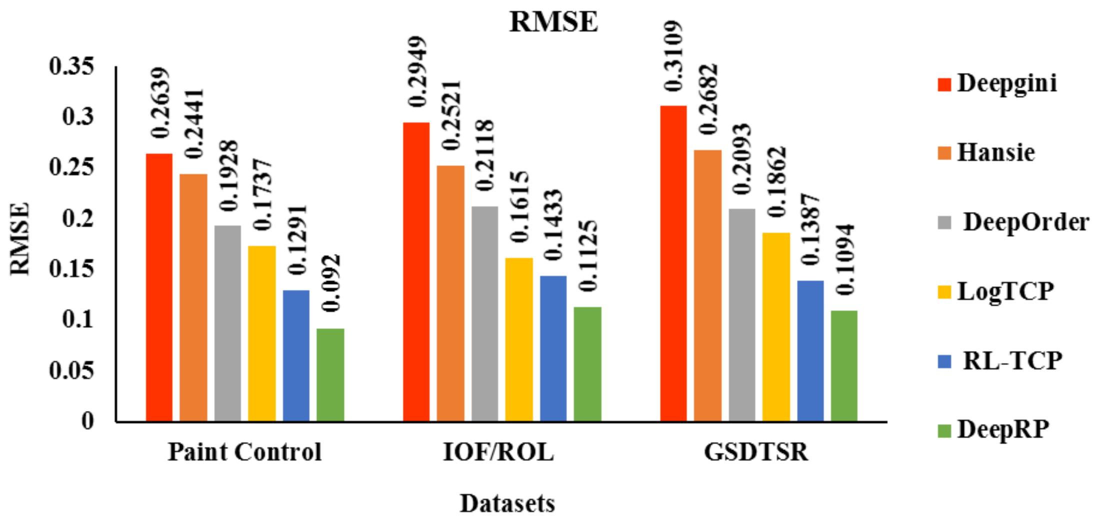

Figure. 9 RMSE Evaluation for given dataset

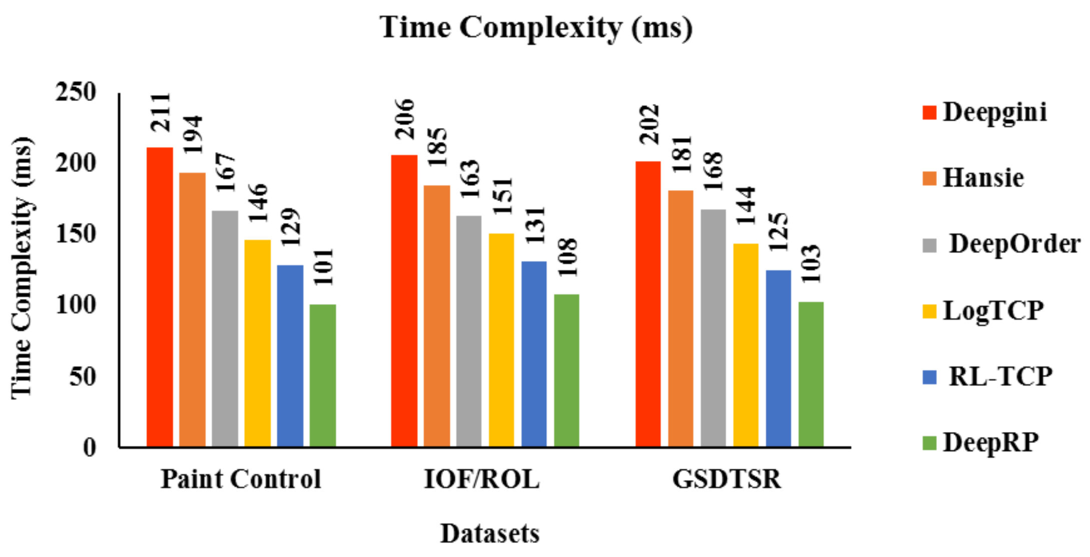

Figure. 10 Comparison of time complexity (ms) for given datasets

DeepRP is 63.46%, 44.07%, 30.77%, 18.06% and 8.97% higher than the Deepgini , Hansie, DeepOrder, LogTCP and RL-TCP models respectively on IOF\ROL dataset.

The Fig. 8 demonstrates the comparison of $APFD_{TA}$ values of proposed and existing models on different TC dataset like Paint Control, IOF/ROL and GSDTSR. It is observed that the DeepRP achieves high $APFD_{TA}$ than other preceding TCP models. For instances, the $APFD_{TA}$ value of DeepRP is 66.67%, 51.79%, 37.09%, 23.19% and 11.84% higher than the Deepgini, Hansie, DeepOrder, LogTCP and RL-TCP models respectively on IOF\ROL dataset.

The Fig. 9 demonstrates the comparison of RMSE values of proposed and existing models on different TC dataset. The RMSE value of DeepRP is 64.81%, 59.21%, 47.73%, 41.25% and 21.12% lesser than Deepgini, Hansie, DeepOrder, LogTCP and RL TCP models respectively on GSDTSR dataset respectively. Form this analysis, it is proved that the proposed model has lesser RMSE values compared to other classical models.

Fig. 10 provides the time complexity analysis for proposed and existing models large tests suites. It is noted that the proposed algorithm can efficiently minimize the time complexities of predicting TCP performance by considering all criteria's compared to other existing algorithms. The DeepRP model decreases about 49%, 43.09%, 38.69%, 28.47% and 17.6% in contrast with the Deepgini, Hansie, DeepOrder, LogTCP and RL-TCP on GSDTSR respectively.

5. Conclusion

In this paper, DeepRP model is proposed to estimate the priority of TCs and increase the accuracy of prioritization in TCP. This method employs DNN to compute the each RL functions such as approximations for reward operations, conversion mechanisms, Q and value operations. $Q -$ Learning determines the agent's further action based on the action-value outcomes. At the end of each episode, the model sorts TCs by therir appropriate results, which are accumulated in a temporary vector, updates the observation, calculates a reward and accumulates the determined score in the vector. Experimental results show DeepRP achieves RMSE values of 0.09, 0.11 and 0.10 on paint control, IOF/ROL and GSDTSR datasets which is lesser than existing models like Deepgini, Hansie, DeepOrder, LogTCP and RL-TCP.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

Conceptualization, methodology, software, validation, Shankar; formal analysis, investigation, Sridhar; resources, data curation, writing—original draft preparation, Shankar; writing—review and editing, Shankar; visualization; supervision, Sridhar.

References

[1] A. Anand and A. Uddin, "Importance of software testing in the process of software development", *International Journal for Scientific Research and Development*, Vol. 12, No. 6, 2019.

[2] P. Kandil, S. Moussa, and N. Badr, N. "A study for regression testing techniques and tools", *International Journal of Soft Computing and Software Engineering*, Vol. 5, No. 4, pp. 64-84, 2015.

[3] M. A. Mascheroni and E. Irrazábal, "Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review", *Computación y Sistemas*, Vol. 22, No. 3, pp. 1009-1038, 2018.

[4] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments", In: *Proc. of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 235-245, 2014.

[5] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey", *Software Testing, Verification and Reliability*, Vol. 22, No. 2, pp. 67-120, 2012.

[6] D. Taneja, R. Singh, A. Singh, and H. Malik, "A Novel technique for test case minimization in object oriented testing", *Procedia Computer Science*, Vol. 167, pp. 2221-2228, 2020.

[7] A. Lawanna and J. Wongwuttiwat, "Test case selection: Vital model for software maintenance", In: *Proc. of 2016 IEEE Region 10 Conference (TENCON)*, pp. 2307-2310, 2016.

[8] H. Wang, M. Yang, L. Jiang, J. Xing, Q. Yang, and F. Yan, "Test case prioritization for service-oriented workflow applications: A perspective of modification impact analysis", *IEEE Access*, Vol. 8, pp. 101260-101273, 2020.

[9] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization", *IEEE Transactions on Software Engineering*, Vol. 33, No. 4, pp. 225-237, 2007.

[10] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization", *Journal of King Saud University-*

Computer and Information Sciences, Vol. 33, No. 9, pp. 1041-1054, 2021.

[11] M. Shahid and S. Ibrahim, "A new code based test case prioritization technique", *International Journal of Software Engineering and Its Applications*, Vol. 8, No. 6, pp. 31-38, 2014.

[12] M. L. M. Shafie, W. M. N. W. Kadir, M. Khatibsyarbini, and M. A. Isa, "Model-based test case prioritization using selective and even-spread count-based methods with scrutinized ordering criterion", *PloS One*, Vol. 15, No. 2, p. e0229312, 2020.

[13] M. L. B. Meyer, "TSAI-Test Selection using Artificial Intelligence for the Support of Continuous Integration", In: *Proc. of 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 306-309, 2021.

[14] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review", *Empirical Software Engineering*, Vol. 27, No. 2, p. 29, 2022.

[15] Z. Wei, H. Wang, I. Ashraf, and W. K. Chan, "Predictive Mutation Analysis of Test Case Prioritization for Deep Neural Networks", In: *Proc. of 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, pp. 682-693, 2022.

[16] M. Bagherzadeh, N. Kahani, and L. Briand, "Reinforcement learning for test case prioritization", *IEEE Transactions on Software Engineering*, Vol. 48, No. 8, pp. 2836-2856, 2022.

[17] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks", In: *ACM Proc. of the 29th International Symposium on Software Testing and Analysis*, pp. 177-188, 2020.

[18] N. Medhat, S. M. Moussa, N. L. Badr, and M. F. Tolba, "A framework for continuous regression and integration testing in IoT systems based on deep learning and search-based techniques", *IEEE Access*, Vol. 8, pp. 215716-215726, 2020.

[19] S. Mondal, and R. Nasre, "Hansie: Hybrid and consensus regression test prioritization", *Journal of Systems and Software*, Vol. 172, pp. 1-42, 2021.

[20] V. Nguyen and B. Le, "RLTCP: a reinforcement learning approach to prioritizing automated user interface tests", *Information and Software Technology*, Vol. 136, pp. 1-16, 2021.

[21] A. Sharif, D. Marijan, and M. Liaaen, "DeepOrder: Deep learning for test case prioritization in continuous integration testing", In: *Proc. of IEEE International Conference on Software Maintenance and Evolution*, pp. 525-534, 2021.

[22] Y. Huang, T. Shu, and Z. Ding, "A learn-to-rank method for model-based regression test case prioritization", *IEEE Access*, Vol. 9, pp. 16365-16382, 2021.

[23] Y. Yang, C. Pan, Z. Li, and R. Zhao, "Adaptive reward computation in reinforcement learning-based continuous integration testing", *IEEE Access*, Vol. 9, pp. 36674-36688, 2021.

[24] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. Briand, "Scalable and accurate test case prioritization in continuous integration contexts", *IEEE Transactions on Software Engineering*, pp. 1-27, 2022.

[25] Z. Chen, J. Chen, W. Wang, J. Zhou, M. Wang, X. Chen, and J. Wang, "Exploring better black-box test case prioritization via log analysis", *ACM Transactions on Software Engineering and Methodology*, pp. 1-33, 2022.

[26] M. Waqar, M. A. Imran, Zaman, M. Muzammal, and J. Kim, "Test suite prioritization based on optimization approach using reinforcement learning", *Applied Sciences*, Vol. 12, No. 13, p. 6772, 2022.

[27] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration", In: *Proc. of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 12-22, 2017.

[28] S. Elbaum, A. Mclaughlin, and J. Penix, *The Google Dataset of Testing Results*, 2014.