Journal of Intelligent Systems and Internet of Things
Vol. 18, No. 02, PP. 361-374, 2026
DOI: https://doi.org/10.54216/JISIoT.180225

Deep Neural Network Graph with Reinforcement Learning for Test Case Prioritization

Shankar Ramakrishnan[1,2,*], E. K. Girisan[3]

[1]Research Scholar, Department of Computer Science, Sri Krishna Adithya College of Arts and Science, Coimbatore, Tamil Nadu, India

[2]Assistant Professor in Department of Computer Technology and Data Science, Sri Krishna Arts and Science College, Coimbatore, Tamil Nadu, India

[3]Associate Professor , Department of Computer Science, Sri Krishna Adithya College of Arts and Science, Coimbatore- 641042 , Tamil Nadu, India

Emails: shankarramakrishnanphd17@gmail.com; ekgirisan@gmail.com

Abstract

Recently, Deep learning (DL) models are increasingly used in Test Case Prioritization (TCP) tasks combining partial and imperfect test case (TC) information into accurate prediction models. Various DL algorithms have been created to improve TC failure prediction and prioritization in CI settings. Among them, Deep Reinforcement Prioritizer (DeepRP) model is developed using Deep Reinforcement Learning (DRL) and Deep Neural Network (DNN) for efficient TCP on huge test suites. But, the model's labelling task is interrupted early, creating difficulty in learning TC features for unlabeled training TCs due to limited resources. To solve this, Deep Graph Reinforcement Prioritizer (DeepGRP) is proposed in this paper to learn the TC features from unlabeled training data for efficient TCP in Regression Testing (RT). In this method, graph neuron stimulation attributes for TCs are created to retrieve the activation graph across DNN layers of DeepRP. The connectivity neuron link defines the activation graph. The proposed deep graph (DG) recognizes the DNN neurons as nodes and the adjacency matrix as the connectivity link among the nodes. Also, the message passing mechanism is applied to aggregate the structural information from the adjacency matrix with neighbouring node features to enhance TCP. By applying this mechanism, DeepGRP captures the high-order dependencies among neurons for efficient activation features which overcomes the traditional activation models and improves the TCP at large scale RT. The DG model prioritizes TCs using Learning-to-Rank (L2R) which learns node attributes from TCs. This enables for better DNN testing efficiency by detecting vulnerabilities early and lower development time for efficient TCP and tackling the difficulty of learning TC characteristics for efficient TCP. Finally, the testing findings suggest that the DeepRP can improve the TCP for large TSs when compared to other common algorithms.

Received: March 19, 2025 Revised: June 12, 2025 Accepted: August 04, 2025

Keywords: Test Case Prioritization; Regression Testing; Deep Graph; Graph-Level Neuron Activation; Learning-to-Rank

1. Introduction

Software Testing (ST) is an important aspect of the software development process since it detects defects and faults in a system and ensures that it works as planned [1]. In ST, Regression Testing (RT) is a crucial process for spotting fresh defects or errors particularly when improving existing systems. It also assures developers that the modified program will not cause new faults in the software testing applications [2].

Continuous Integration (CI) is now widely used in software projects. It streamlines program releases and incorporates RT frequently [3, 4]. All TCs are challenging to run, nevertheless, depending on time, money and

resource limits. Furthermore, the fast delivery intervals for changed software raise the time utilization, therefore restricting the RT efficacy. In software, RT defines the operating expenditures and case resources by utilizing the methods like minimization, selection and prioritizing [5, 6]. While minimization [7] removes irrelevant TCs and selection [8] chooses the most important ones for software testing. By reordering the TS to find the optimal sequence of TCs, TCP models [9] improve goals including pre-mature failure detection. These approaches assist to lower test sets depending on certain criteria.

Among these three TCP approaches, the software sector makes great use of TCP ones to improve RT [10] efficiency. By means of overlapping software TC testing and troubleshooting, they reduce testing costs. Initially, Although TCP prioritizes the most important TCs, it permits continuous testing until either resources are exhausted or all TCs have been executed. TCP is crucial for streamlining software development and making the most of analyzing resources [11, 12]. It may be challenging to prioritize TCs, however, due to the abundance of TCs, unpredictable changes in requirements and complex relationships. Moreover, it will be easier to spot issues in the RT if significant TCs are consistently given considerable scrutiny.

Artificial intelligence (AI) like Machine Learning (ML) and DL has considerably lowered software failure rates by simplifying processes in software engineering, notably, which may assist address, the TCP issue. DL-based TCP techniques estimate TC priority in large TSs utilizing TC duration and history execution status [15, 16] unlike ML models. One of the sophisticated DL models, Reinforcement Learning (RL) showed great promise in regression testing by continuously changing prioritizing techniques depending on test comments [17, 18]. For example, Different ranking techniques help to simulate the sequential links between the CI platform and the TCP into the RL issue [19]. Excellent ranking reliability for TCs regression involves the RL models could continually and automatically demonstrate the TCP method. Nonetheless, lack scalability, inability to handle large-scale TCs and difficulty modifying and optimizing hyperactive parameters define RL-based algorithms as problematic. Furthermore, DNN techniques cannot retain significant information for longer, therefore the network architecture becomes complicated.

Deep Reinforcement Prioritizer (DeepRP) is developed in this article to solve the above-mentioned issues. This model uses DRL to train additional TC information, including modifications to source code, revision regulation and code coverage, thus improving the self-optimization and flexible capacity for TCP. Value operation, Q function, transformation system and reward function are among the many RL functions approximatively modelled in DRL training by DNN. A RL system called Q-Learning bases agent behavior on TC attributes as input and priority as output. It entails updating observations, organizing TCs, calculating incentives and maintaining particular marks. The disparity among allocated and ideal rankings determines the incentive to improve TC efficiency.

DeepGRP model is then proposed to learn the TC features for unlabeled training TCs for efficient TCP in RT. In this method, Graph-level neuron activation characteristics for TCs are used in this technique to derive the activation graph among the DNN layers. The neural connection relationship is the definition of the activation graph. While certain finer-grained properties, such as the activation of the neuron, have been recovered by the existing model activation-based prioritization algorithms as they neglect other parameters that are of more hidden layer. Consequently, the suggested DNN views the adjacency matrix as the connection relationship between the nodes, including the DNN neurons. In order to get integrated node characteristics, the adjacency matrix and node attributes are sent together via messages. These features include nearby node properties and structural data among nodes, which may be extremely beneficial for TCP. By collecting finest activation features from TCs and converting model activations into spatial neuron connections, the DG overcomes limitations based on model activation methodologies. By learning one kind of scenario, it may use for multiple TCP, making it more comprehensive than model activation-based techniques. This DG model uses L2R to rank test instances by learning node attributes. Early vulnerability detection reduces processing time for efficient TCP and resolves learning TC attribute complexity, increasing the effectiveness of DNN testing.

The following sets the stage for the remainder of the article: In Section II, the researchers describe their findings on TCP prediction using DL algorithms. Section III describes the proposed approach, and Section IV compares its results to those of the existing algorithms. In Section V, we provide the study's findings and suggestions for enhancements.

2. Literature Survey

Huang et al. [20] suggested a sorting mechanism using the Enlarged Finite State Machine (EFSM) for TCP testing. This method includes random forest with heuristic prioritization techniques to rectify the fault security level and time cost of TC implementation potentially affecting construct validity. Nevertheless, it generates high temporal computational complexity.Yang et al. [21] developed two TCP active descending frame models for TCP. Initially, a fixed-size sliding window was used to allocate a pre-set duration of recent historical data for all CI tests. After

that, dynamic sliding window approaches were created, in which the size of the window was continually adaptable to all CI tests. However, in many TC iterations provided delayed learning or convoluted convergence due to ineffective reward levels.

Pan et al. [22] developed a DNN for TCP based on Gini impurity. This system organizes TCs using metric values and Gini impurity formula, which produces reliant functions into a guided acyclic network. This TCP filters out TCs, fed into DNN, and ensures data feature engineering was performed appropriately for the TCP. However, this model only considered independent unit TCs, requiring the use of more advanced models.Yaraghi et al. [23] devised a conceptual data model to facilitate CI for data resources and their associations. Then, a complete set of characteristics including all features previously used in related investigations was defined using this data model. Moreover, these features were applied to train machine-learning models and accurately prioritize TCs. On the other hand, this model necessitates high-quality datasets to evaluate TCP models.

Chen et al. [24] developed a LogTCP, which constitutes log pre-analysis, log depiction and TCP components. This model states that by fusing various log exemplification techniques with various prioritization strategies and several log-based schemes were built using the conceptualization of LogTCP. However, this approach results in high time complexity issues.Waqar et al. [25] developed TS prioritization by combining an optimization technique with an RL model. Initially, the administration log files were compiled by employing the behavior tracing models. Then, RL model was applied on the gathered log data to establish the possible future beneficial rewards. Finally, the error seeding approach was employed to check the performance from the software specialists. However, total reward activities were more time-consuming than their partial reward counterparts were.

Abdelkarim & ElAdawi, [26] constructed a TCP Using End-to-End DNN (TCP-Net++). TC metadata, coverage details and failure records are among the characteristics that this model incorporates to discover high-dimensional connection between source code and TCs. It prioritizes failure frequencies and coverage criteria for real-life industrial software packages, but results are not statistically verified due to the use of existing datasets.Chen et al. [27] developed a DNN model to prioritize the teat case using the activation graph (Actgraph). The adjacency matrix represents the connections between the DNN neurons, which are identified by the algorithm as nodes. Through message forwarding, it gathers neighbor node attributes and structural data, which might be used for TCP. Node features and the adjacency matrix were then combined. But, this model results with high time complexity issues.
Manikkannan & Babu, [28] presented a TCP for Software Superiority Assertion that uses an Embedded Auto Encoder (EAE) for TCP. To reduce noise, each benchmark's code reportage was accessed using the resource code repositories. The data was then fed into the EAE, which prioritizes tasks and reduces phases to produce high-quality software free from defects. However, more features like source code modifications were needed to enhance efficiency.

In comparison to coverage-based approaches, DL-based approaches prioritize test cases more efficiently and effectively, according to a statistical view of DNN. It typically takes a lot of time and effort to label testing in the actual world because they do not have labels. In earlier models, the labelling was better. However, because to limitations in resources, the tagging process is abruptly stopped at an arbitrary moment. To label with little resource use, the TCP approaches are necessary. Achieving maximum advantage is possible with such a priority strategy. The model proposed in this study learn the test case features for unlabelled training test cases.

3. Proposed Methodology

Figure 1. depicts the entire structure of the proposed DeepGRP model.

3.1 Preliminaries

A proposed TCP method DeepGRP is mainly based on the model activation graph task. The three phases involved in this model are (i) TC Impulsation (ii) Feature Extraction by DNN DeepRP and (iii) Ranking Model Construction. Emphasizing its dynamic features, activation graph models have turned DNN into graphs. Though they do not provide TC stimulation data, in which the undirect weighted network with neurons are considered as nodes and model weights as edges. Rather, a guided weighted graph with model weights as the skeleton and authorization values as the graph is employed.

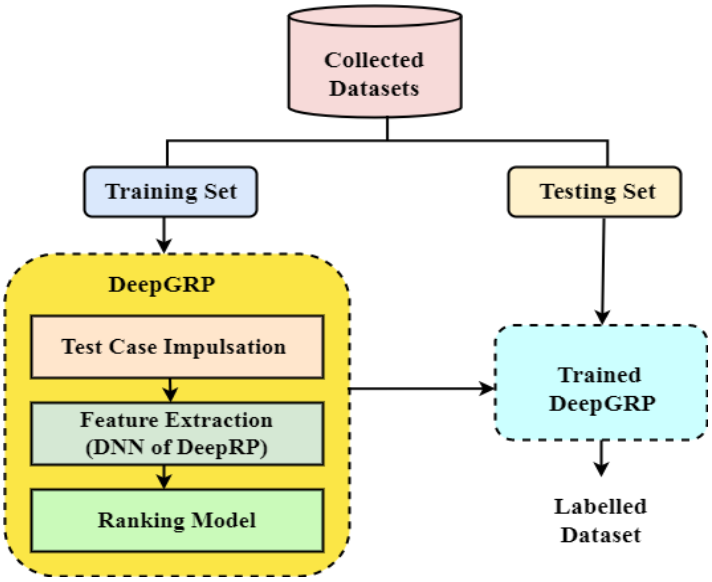

Figure 1. Pipeline of the proposed model

3.2 Test Case Prioritization using Deep Reinforcement Learning

In RT, prioritizing the TC on large TSs is important to eliminate the loss of information issues. So, DeepRP [20] model is proposed for TCP on large TSs. DeepRP uses a DRL and DNN. In proposed DeepGRP, the concept of DNN from DeepRP is used for feature extraction to learn the TC features for unlabeled training TC for efficient TCP.

3.2.1 Test Case Impulsation (TCI)

During testing, the proposed DeepGRP uses a TCP paradigm based on model activation. In TCI, the trained DNN's layers provide initialization values in response to the TCs they receive. The process of building graphs involves aggregating and standardizing the activations and weights of all neurons. The TC a is created by training the DNN with layers l with x^{th} neuron the L^{th} layer as N_x^L. In TC, a^{th} will be fed into DNN and acquire an output neuron layer in the DNN. The activation value $\Phi_x^L$and N_x^L is computed in Eq. (1),

$$\Phi_x^L(a) = \frac{1}{H_L * W_L} \sum_{H_L} \sum_{W_L} \mathcal{F}_x^L(a) \quad (1)$$

where $\mathcal{F}_x^L(a) \in \mathbb{R}^{H_L * W_L * C_L}$is the attribute output map of N_x^L when i is input. H, W and C represents the height, width and channels respectively with respect to neuron layers L. In case of the convolutional layer, the output dimension of $H_L * W_L$with the fully connected (FC) dimension will be $1 * 1$.

In order to normalize the neuron stimulation integer $\Phi L\ (a)$ within the range [0,1], the max-min normalization is triggered by computing it in Eq. (2) for all layers.

$$\Phi^L(a) = \frac{\Phi^L(a) - \min(\Phi^L(a))}{(max\ \Phi^L(a)) - (min \Phi^L(a))} \quad (2)$$

Eq. (3) depicts the neuron weight of N_x^L,

$$w_{y,x}^{L-1,L} = \frac{1}{H_L^\theta * W_L^\theta} \sum_{H_L^\theta} \sum_{W_L^\theta} \theta_{x,y}^{L-1,L} \quad (3)$$

The weight variable among the neuron N_y^{L-1} and N_x^L will be slightly indicated to $\theta_{x,y}^{L-1,L} \in \mathbb{R}^{H_L^\theta * W_L^\theta}$. The weight dimension of FC layer is 1*1 in which the dimension of convolutional layer weight L[th] will be $H_L^\theta * W_L^\theta$. In Eq. (4), neuron weight $w_{x,y}^{L-1,L}$ of each layer is standardized and transform $w_{x,y}^{L-1,L}$) of each layer within the range [0,1].

$$w_{x,y}^{L-1,L} = \frac{w_{x,y}^{L-1,L} - \min(w_{x,y}^{L-1,L})}{\max(w_{x,y}^{L-1,L}) - \min(w_{x,y}^{L-1,L})} \quad (4)$$

In order to save computation costs, DeepGRP retrieves the neuron stimulation and weights of the last m layers of the DNN.

3.2.2 Feature Extraction

The proposed method involves extracting the activation graph between DNN layers using graph-level neuron activation features for TCs. The suggested graph-level neuron activation differs from ActGraph [27] and other DNN-based TCP approaches in several ways. Models as if ActGraph [27] constructs activation graphs by directly linking neuron connections with adjacency matrices highly suffers from high computational cost and limited scalability on large dataset. In addition, ActGraph primarily captures local neuron activations and immediate connectivity overlooking higher-order dependencies among neurons. In contrast, the proposed DeepGRP applies graph-level aggregation with message passing, which consolidates both activation values and structural node features from neighboring layers. Additionally, graph-level neuron activation enables higher-order dependencies by propagating information across multiple layers of the activation graph which highly presents the global structural patterns that are not captured by simple layer-wise activation. This reduces computational overhead, captures richer high-level structural relationships between the activation features and improves prioritization accuracy. Additionally, earlier DNN-based TCP methods handles test cases as isolated units, DeepGRP generalizes across unlabeled test cases by leveraging structural graph properties, making it more effective for large-scale regression testing.

In this model, stimulation graphs using DNN activations and node characteristics taken from the contiguity matrix are developed as the part of the feature extraction. By gathering features from the stimulation values of TCs, DeepGRP generates a grading model for each layer of the DNN model using the L2R architecture. The technique uses weighted edges to create dispersion variations among test instances, aiming to capture the neuron properties in a challenging manner. Priority TC from the stimulation graph are assisted using the Graph Neural Network (GNN) to improve the node characteristics efficacy. In DNN, initiation values are transmitted from the previous layer to the next layer, gathering the properties of current and neighboring nodes. The directed activation network is used to extract weighted in-degree node attributes to emphasize the importance of nodes. To highlight the significance of nodes, weighted in-degree node properties are extracted using the influenced activation network. Higher-order node characteristics with the centre node attributes are obtained by aggregating the proximity matrix. DNN is aligned as the directed weighted graph using the stimulation values as the bases for data drift. In accordance to Eq. (5), it is determined that d = (v, e), where v and e denotes the neuron set and directed weighted edges.

$$X_{y,x} = \begin{cases} w_{y,x} * \Phi_x & , \quad V_y \in \Pi_d^-(V_x) \\ 0 & , \quad V_y \notin \Pi_d^-(V_x) \end{cases} \tag{5}$$

Where X represents the neighbouring matrix of d, V_x will be the x^{th} node of d, $w_{x,y}$ is the weight among V_y and V_x and $\Pi_d^-(V_x) = \{V_y | V_y \in v(d) \wedge \langle V_y, V_x \rangle \in e(d) \wedge V_y \neq V_x\}$ will be adjusted as processor in V_x. The weighted in-degree is employed as the node attributes ($\mathcal{N}_f$). The easiest approach to evaluate the degree property is the way to capture links among the nodes. Since the weights of neighboring input edges are thought to be comparable to the weight of V_x in Eq. (6), the weight of V_x is determined by adding them.

$$\mathcal{N}_f = \sum_y X_{y,x}, V_y \in \Pi_d^-(V_x) \tag{6}$$

In this case, $\mathcal{N}_f$ represents the node attributes of V_x. Features with lesser complexity degree will be weighted high. Eq. (7) states that the center node feature (⟦CN _f) is generated by combining the activation graph's node aspects and adjacent matrix during the GNN communication task.

$$\mathcal{CN}_f = \Psi(X, \mathcal{N}_f) \tag{7}$$

Where, averaging operation $\Psi(.)$can employ the $Add(.), Max(.)$ and $Average(.)$. For the proposed TCP model, $Add(.)$ is executed. The proposed model calculates the $\mathcal{CN}_f$ of the last two layers after executing all nodes, aiming for deeper activation to fully express high-dimensional TC characteristics. Since the multiple-stage $\mathcal{CN}_f$ needs a minimum of four weight and activation layers, it set to $m = 4$.

Since, the last m = 4 layers are used in the DNN model, feature extraction focuses on the most informative and task-relevant representations. In DNN, these final (deeper) layers capture the high-level abstractions that are directly relevant for TCP and for distinguishing fault-prone test cases. Whereas, the earlier layers mainly capture the low-level patterns provides lower contribution on pritorization. Extracting activations from all layers adds substantial computational cost while using fewer than four layers reduces the pritorization efficiency. Empirical evaluation further demonstrated that m = 4 provides the best trade-off between accuracy and inference time making it well-suited for CI environments.

3.3 Constructions of Ranking Model

To build an L2R-based ranking model, DeepGRP employs the Extreme Boost (EB) model that adapts the dispersed gradient RL model. In this model, EB model is adopted as the L2R method due to its superior scalability and efficiency compared to other ranking algorithms like LambdaMART, RankNet and ListNet. Specifically, EB employs histogram based tree construction, parallel learning and build-in regularization allowing it to effectively handle the large-scale datasets like GSDTSR. In contrast, LambdaMART requires more complex pairwise or listwise transformations and were often more sensitive to noisy or partially labeled data, which are common in CI environments. Moreover, EB integrates effectively with the graph-level activation features extracted by DeepGRP enabling effective training and fast inference, making it suitable for deployment in real-time RT pipelines.

The EB ranking model is trained using the test set's $\mathcal{CN}_{\oint}$ and the DNN's prediction is marked as 0 (right) or 1 (wrong) according to the instances accuracy.

$$\boldsymbol{O}\ (\mathcal{CN}_{\oint}, b) = \boldsymbol{\mathcal{L}}\left(b, \hat{b}\right) + \sum_{t=1}^{\mathcal{T}} \Omega(\mathcal{F}_t) \tag{8}$$

Where, $\hat{b} = \sum_{t=1}^{\mathcal{T}} \mathcal{F}_t\left(\mathcal{CN}_{\oint}\right)$, $\mathcal{F}_t(\mathcal{CN}_{\oint})$ represents an identified rate of t^{th} tree where b is determined to be 0 or 1.

3.3.1 Practicality Inspection of Center Node Feature

The $\mathcal{CN}_{\oint}$ are determined by message transferring accumulation and defines the value of m in the proposed DeepGRP model. Eq. (9) and (10) determines the activation value ∂ and weight $\mathcal{W}$ of each layer in DNN with N neurons which are by case a initiating the resultant value of each layer.

$$\partial = [\partial_0(a)\partial_1(a) \ldots \partial_{n-1}(a)]_{1*n} \tag{9}$$

$$\boldsymbol{\mathcal{W}} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n-1} \\ w_{1,0} & w_{1,1} & \cdots & w_{0,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ w_{n-1,0} & w_{n-1,1} & \cdots & w_{n-1,n-1} \end{bmatrix}_{n*n} \tag{10}$$

Eq. (2) and (4), indicates the standardization of layers. As seen in Eq. (11), the adjacency matrix is computed concurrently in accordance with Eq. (5).

$$\boldsymbol{\mathcal{W}} = \begin{bmatrix} \partial_0.w_{0,0} & \partial_0.w_{0,1} & \cdots & \partial_0.w_{0,n-1} \\ \partial_0.w_{1,0} & \partial_0.w_{1,1} & \cdots & \partial_0.w_{0,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ \partial_0.w_{n-1,0} & \partial_0.w_{n-1,1} & \cdots & \partial_0.w_{n-1,n-1} \end{bmatrix}_{n*n} \tag{11}$$

Then, the $\mathcal{N}_{\oint}$ is evaluated in Eq. (12),

$$\mathcal{N}_{\oint} = [\partial_0 \sum^{y} w_{y,0} \ \cdots \partial_{n-1} \sum^{y} w_{y,n-1}\ ]_{1*n} \tag{12}$$

Where, the stimulation value and input edges of V_x are averaged to yield the node attribute $\mathcal{N}_{\oint}\ (x)$ indicated as $\mathcal{N}_{\oint}\ (x)$ is $\partial_0 \sum^{y} w_{y,x}$ Finally, $\mathcal{CN}_{\oint}$ is analyzed using X and $\mathcal{N}_{\oint}$ as in Eq. (13),

$$\mathcal{CN}_f = [\sum^{q} \partial_q w_{q,0}\, \mathcal{N}_{\oint}(q) \ \cdots \sum^{q} \partial_q w_{q,n-1}\, \mathcal{N}_{\oint}(q)]_{1*n} \tag{13}$$

Where, $\mathcal{CN}_f(x)$ of V_x is $\sum^{q} \partial_q w_{q,x}\, \mathcal{N}_f(q)$. Typically, $\mathcal{CN}_f(x)$

Where, $\mathcal{CN}_f(x)$ of V_x is $\sum^{q} \partial_q w_{q,x}\, \mathcal{N}_f(q)$ is produced from activation values by considering the $\mathcal{N}_f$ $n$eurons within higher layer of V_x. DeepGRP aggregates three network layers for analytical $\mathcal{CN}_f$ and adopts the latest four layers of DNN for effective $\mathcal{CN}_f$.

The Figure 2. and below algorithm illustrates the framework of DeepGRP.

Algorithm: DeepGRP

Input: *DNN $\mathcal{F}_1$ to be validated; The final m layers are adopted; Testing Dataset $a_G \in A = \{a_1, a_2, .., a_n\}$, the group of $\mathcal{CN}_f$*

Output: *Ranking Model $\mathcal{F}_2$ for TCP*

1. for L in m do
2. Calculate the neuron weight $w^{L-1,L}$ as specified in Eq. (3)
3. Regularize the $w^{L-1,L}$ using Eq. (4)
4. end for

5. $\mathcal{CN}_f = \{\theta\}$
6. for a_G in A do
7. for L in m do
8. Execute Neuron Activation $\partial^1(a_G)$ utilizing the Eq. (1)
9. Define $\partial^1(a_G)$ by using the Eq. (2)
10. end for
11. Initialize the directed weighted graph d_G
12. Obtain the Adjacency matrix X_G as in Eq. (5)
13. Update the $\mathcal{N}_f(G)$ using Eq. (6)
14. Calculate the $\mathcal{CN}_f(G)$ using Eq. (7)
15. $\mathcal{CN}_f \leftarrow \mathcal{N}_f(G) \cup \mathcal{CN}_f(G)$
16. $end\ for$
17. Train the ranking model using Eq. (8)
18. Return $\mathcal{F}_2$

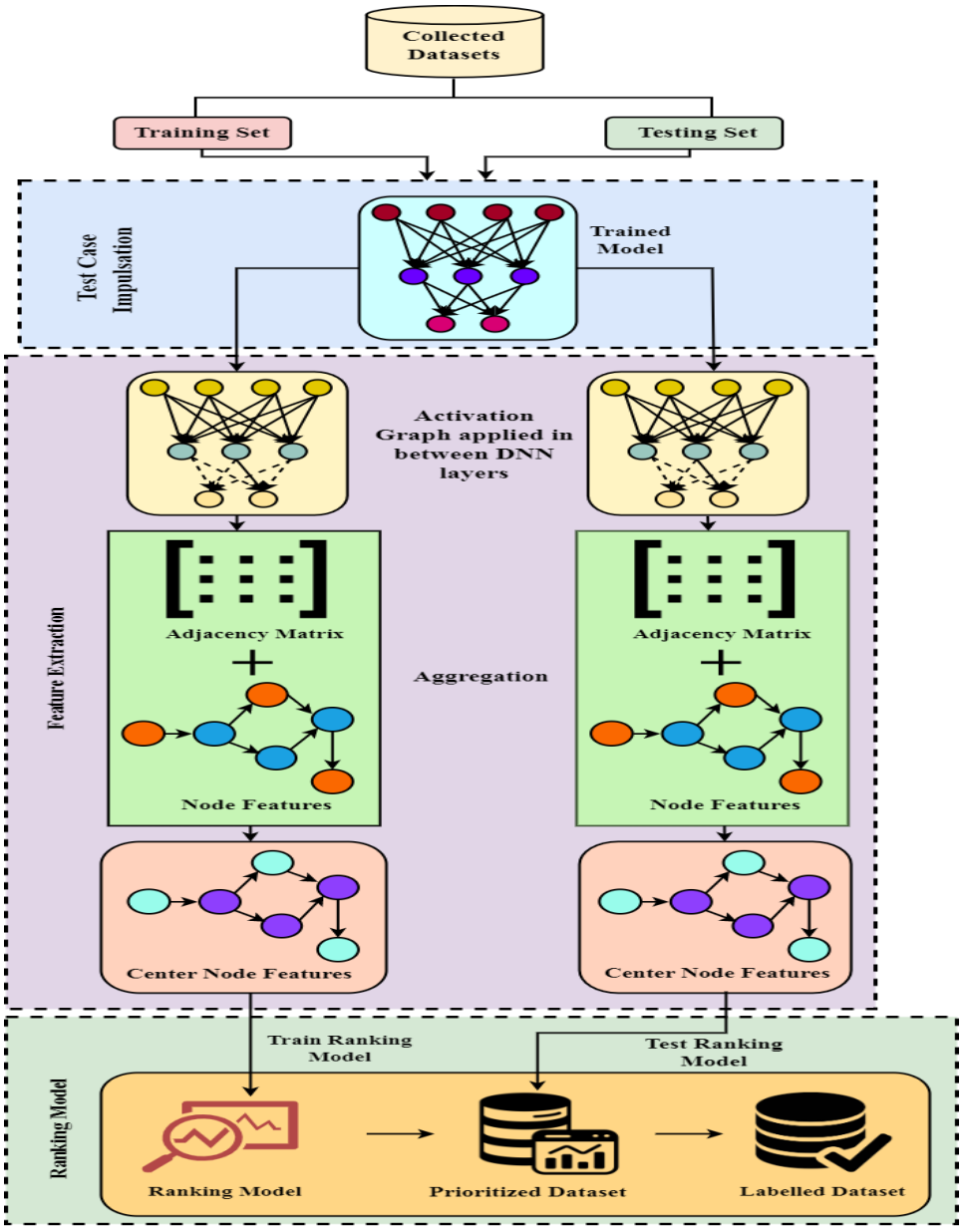

Figure 2. Structure of DeepGRP

3.4 Deployment in Cloud based IoT Testing Environments

DeepGRP could be deployed in cloud-based IoT testing environments by integrating it into continuous integration pipelines for automatic TCP across large-scale IoT testing environments. The graph-based feature extraction enables handling of unlabeled test data from diverse IoT devices while the ranking model ensures critical faults are detected early. Leveraging the scalability in cloud platforms, DeepGRP can process high volume IoT test executions in parallel, thereby reducing regression testing costs and enhancing reliability in dynamic IoT systems.

4. Simulation Results

4.1 Dataset Description

For the experimental purposes, Google Shared Dataset of TS results (GSDTSR) [30], Paint Control [29], and IOF/ROL [29] are examples of industrial databases that are utilized. In respect to the CI environment, the data sets in all case studies include past test implementation and evaluation data are analyzed across 300 CI iterations. The structure of the data sets is shown in Table 1.

Table 1: Gathered Information form the Datasets

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

"TCs" denotes the total number of test cases available in each dataset n Table 1., while "CI Cycles" refers to the number of continuous integration iterations during which those test cases were executed. The "Verdicts" represents the total number of test execution outcomes (pass or fail) generated across all CI cycles for each datasets. For example, in the Paint Control dataset, 114 test cases executed over 312 CI cycles produced 25,594 verdicts. Similarly, the IOF\ROL datasets accumulated their verdict counts based on the number of test cases executed per cycle. These verdicts evaluate fault detection performance, as the proposed model prioritizes test cases with a higher likelihood of producing failed verdicts. The "Failed" column indicates the proportion of these verdicts that resulted in failure, thereby reflecting the fault-proneness of each dataset.

The aforementioned datasets are all verified and categorized based on implementation duration, time update duration and most recent outcome. It is important to note that these datasets include only offer past execution details of TC. Since, the Paint Control dataset contains only 114 test cases, which is too small to represent the scalability challenges of real-world regression testing. Such a limited dataset risks overfitting, provides less fault diversity and reduces statistical reliability. Therefore, it is useful for preliminary evaluation and results on Paint Control must be interpreted cautiously validated against larger datasets such as IOF/ROL and GSDTSR.

4.2 Performance Evaluation Metrics

In order to efficient of the proposed DeepGRP model, it is compared with the existing algorithms like LogTCP [24], TCP-Net++ [26], Actgraph [27] and TCP-EAE [28]. The proposed and existing models were executed in Python 3.7.8 using the dataset illustrated in section 4.1. Table 2. depicts the hyperparameter for the proposed model.

Table 2: Hyperparameter Settings

Models	**Hyperparameters**	**Values**
EB Ranking Model	n-estimators	300
	Maximum depth	6
	Child weight	1
	Early stopping rounds	30
	Learning Rate	0.0002
DNN	Hidden layers	4 (last m layers used)
	Units per layer	512

	Batch Size	256
	Activation function	ReLU
	Learning Rate	0.0001
GNN	Layers (Message Passing)	2
	Hidden Dimension	256
DRL	Discount factor (γ)	0.95
	ε-greedy schedule	1.0 → 0.05 (30k steps)
	Replay buffer size	1e5
	Batch size	256
	Target update	Every 2000 steps

The metrics exploited to evaluate proposed and existing models are provided below.

Average percentage of faults detected (APFD)

It influences the TS's speed at identifying errors. APFD determines the weighted mean of the mistakes found throughout TS run.

$$APFD = 1 - (\frac{TF_1 + TF_2 + \cdots + TF_m}{NM} + \frac{1}{2n}) \quad (14)$$

In Eq. (14), m is the aggregate number of errors found in the program for TC execution and T is the resultant TS. n is the total TC number, $Tf_1, Tf_2, \ldots, Tf_m$ are the first T points that disclose the m.

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

In order to execute TS T', the usual percentage of cost-effective TCs is compared to the typical percentage of fault severity in order to get the weighted average proportion of mistakes.

$$APFD_c = \frac{\sum_{i=1}^{m}(f_i * (\sum_{j=TF_i}^{n} t_j - \frac{1}{2} tTF_i))}{\sum_{j=TF_i}^{n} t_j * \sum_{i=1}^{m}(f_i)} \quad (15)$$

In Eq. (15), T represents the TS including n TCs with overheads $t_1, t_2, \ldots \ldots, t_n$. F be the set of m mistakes revealed in T; the degrees of those mistakes f_1, f_2, ….., f_m. TF_i as the initial TC that identifies the issue i is TF_i.

Normalized Average Percentage of Faults Detected (NAPFD)

It integrates both the fault data and time detection to calculate TCP performance. It is signified in Eq. (16),

$$NAPFD = p - (\frac{TF_1 + TF_2 + \cdots + TF_m)}{m * n} + \frac{p}{2n}) \quad (16)$$

In Eq. (16), p is calculated by dividing the identified faults to the prioritized TS of aggregate number of detected error. TF_i is the number of the TC in which fault i is determined by testing the significance order. When no error is determined, TF_i is set to 0.

Average Percentage of Faults Detected ($APFD_a$)

$APFD_a$ is an enhanced version of $APFD_c$ which constitutes TC process for the testing task. The $APFD_a$ notation is provided in Eq. (17),

$$APFD_a = \left(1 - \sum_{i=1}^{m} \frac{\sum_{j=1}^{TF_i} C_j}{m \sum_{j=1}^{n} C_j}\right) * 100\% \quad (17)$$

In above Eq. (17), C_j is the cost obtained in the j^{th} TC.

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

It is the particular case of $APFD_c$ for sample instances with same test costs and mistake challenges.

$$APFD_{TA} = \frac{\sum_{i=1}^{m}(\sum_{j=TF_i}^{n} C_j - \frac{1}{2} CTF_i))}{\sum_{j=1}^{n} C_j * |\sigma|} \quad (18)$$

In Eq. (18), σ is constant among the first and last TCs in the whole TS.

Root mean square error (RMSE)

It determines the variation among the observed and estimated NAPFD values. Using the dissimilar value calculated for T' in a CI repetition specified by learning model ($\hat{u}_C$), the adjoint estimated value T' is used by RL model.

$$RMSE\ (\Psi) = \sqrt{\frac{\sum_q^{CI}(\hat{u}_q - u_q)}{CI}} \tag{19}$$

In Eq. (19), CI is the total system cycles. Lesser RMSE determines more precise results. $\hat{u}_q$ defines the best prioritizing identified by RL for an ideal priority task.

Figure 3. demonstrates the APFD analysis for suggested and existing models on different TC dataset. DeepGRP models clearly show more efficiency than conventional models for TCP on large test sets. For instance, APFD value of DeepGRP is 46.77%, 35.82%, 26.39%, 16.67% and 7.06% higher than the LogTCP, TCP-Net++, Actgraph, TCP-EAE and DeepRP models respectively on GSDTSR dataset. Figure 4. shows that the suggested model attained higher $APFD_c$ than other methods. For example, $APFD_c$ of DeepGRP is 60%, 37.5%, 24.71%, 15.79% and 6.02% greater than the LogTCP, TCP-Net++, Actgraph, TCP-EAE and DeepRP models respectively on Paint Control dataset.

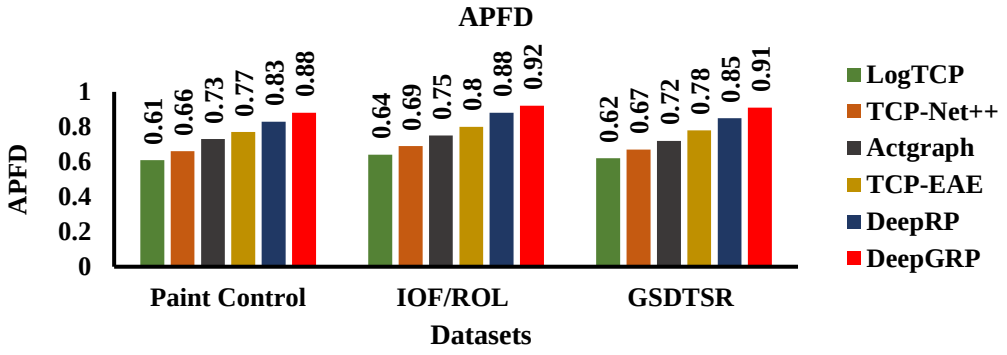

Figure 3. APFD Evaluation for Proposed and Existing Models

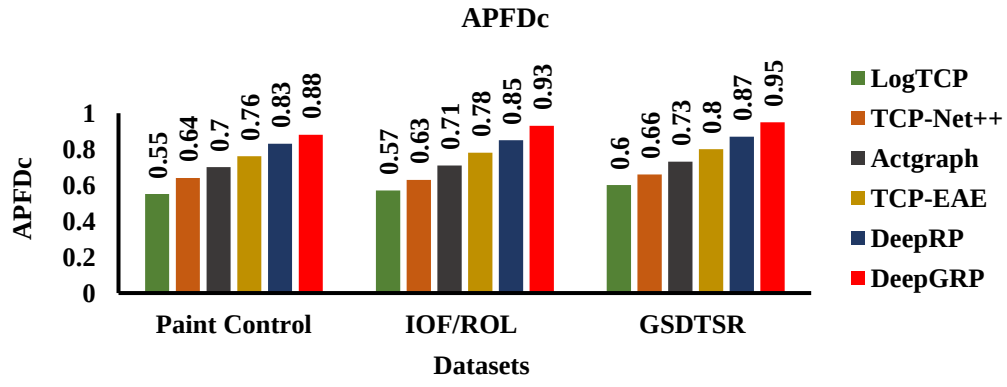

Figure 4. $APFD_c$ Analysis for various methods

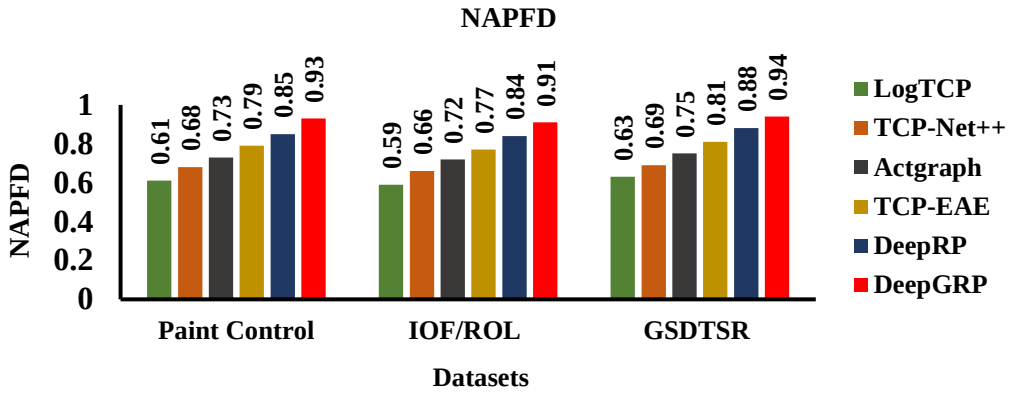

Figure 5. Evaluation of NAPFD Metric for different models

Figure 5. shows on many TC datasets the NAPFD assessment of produced and accessible models. For instances, NAPFD value of DeepGRP is 54.24%, 37.88%, 26.39%, 18.18% and 8.33% higher than

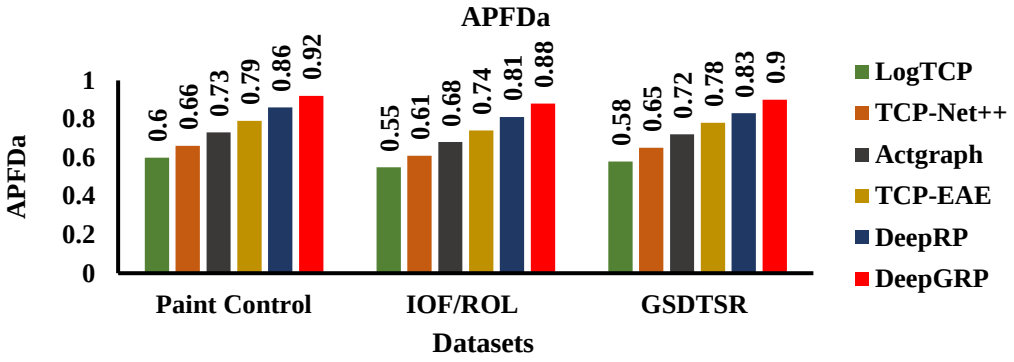

Figure 6. $APFD_a$ Analysis for Suggested and Traditional Models

The LogTCP, TCP-Net++, Actgraph, TCP-EAE and DeepRP models on IOF/ROL dataset correspondingly.Figure 6. shows evaluation of $APFD_a$ of developed and classical methods on different dataset. $APFD_a$ value of DeepGRP is 55.17%, 38.46%, 25%, 15.38% and 8.43% higher than the LogTCP, TCP-Net++, Actgraph, TCP-EAE and DeepRP models respectively on GSDTSR dataset. In both the comparison, the proposed DeepGRP models achieves high NAPFD and $APFD_a$ results than other existing models.

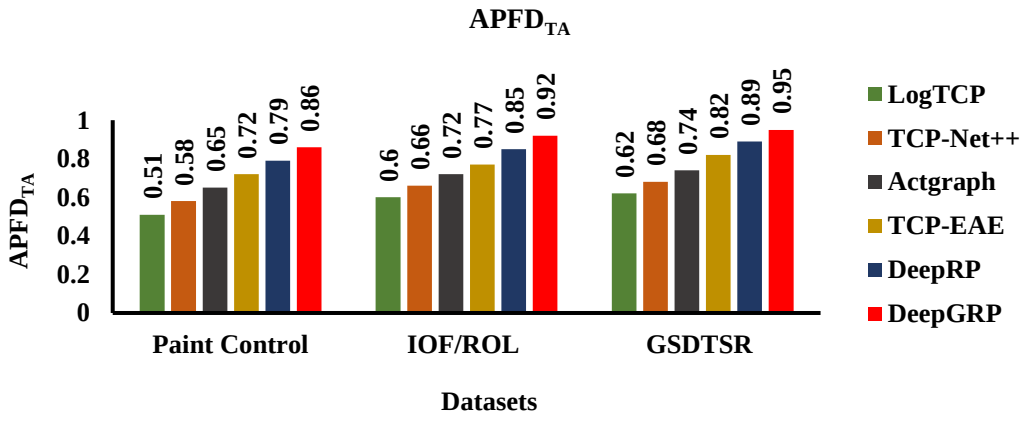

Figure 7. $APFD_{TA}$ Evaluation for Proposed and Existing models

Using a variety of TC datasets, Figure 7. Compare values for traditional and proposed models. es the$APFD_{TA}$This examination demonstrates that, in comparison to other TCP models, DeepGRP attains a high $APFD_{TA}$. On the Paint Control dataset, for example, the $APFD_{TA}$ of DeepGRP is 68.63% greater than that of LogTCP, 45.28% higher than TCP-Net++, 32.31% higher than Actgraph, 19.44% higher than TCP-EAE, and 8.86% higher than DeepRP.

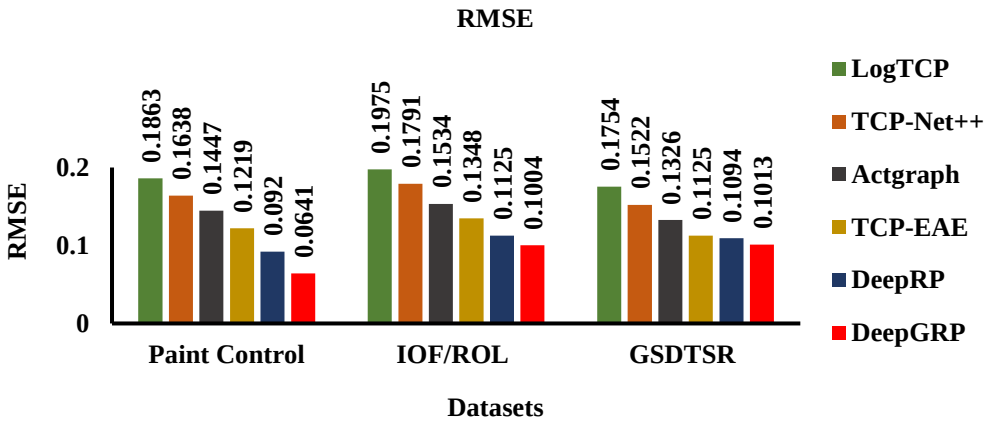

Figure 8. RMSE Evaluation for Proposed and Existing models

The RMSE assessment of both current and proposed models on various TC datasets is shown in Figure 8. On the IOF/ROL dataset, DeepGRP outperforms the other models with RMSE values that are 49.52%, 43.94%, 34.55%, 25.52%, and 10.75% higher, respectively. Based on these results, it is clear that the proposed model outperforms more traditional approaches in terms of RMSE values.

Table 3: P-value Analysis for Proposed and Existing Models

Dataset	**LogTCP**	**TCP-Net++**	**Actgraph**	**DeepRP**	**DeepGRP**
Paint Control	2.41×10^{-4}	3.27×10^{-5}	7.19×10^{-6}	5.88×10^{-5}	$\mathbf{1.82\times10^{-6}}$
IOF\/ROL	1.13×10^{-3}	2.96×10^{-4}	4.51×10^{-5}	2.14×10^{-5}	$\mathbf{7.33\times10^{-6}}$
GSDTSR	8.62×10^{-4}	1.75×10^{-4}	3.97×10^{-5}	2.85×10^{-5}	$\mathbf{8.31\times10^{-6}}$

Table 3. presents the p-value analysis for the proposed and existing models om all datasets. In this analysis, the proposed model provides lower p-values compared to TCP-Net++, Actgraph, DeepRP models. This indicates that the DeepGRP model has statistically stronger evidence against the null hypothesis, thereby confirming the reliability of its improvements in TCP effectiveness. However, han LogTCP produces even smaller p-values on these datasets. This is largely due to its simplified log-based heuristics, which make statistical variations easier to capture in limited or less complex datasets. Despite this, such heuristics lack robustness and generalization leading to weaker performance on larger RT scenarios. By contrast, DeepGRP leverages graph-based activation features and message passing to consistently deliver statistically significant improvements across varied datasets, balancing both accuracy and scalability.

Table 4: Inference Time (s) for Proposed and Existing Models

Dataset	**LogTCP**	**TCP-Net++**	**Actgraph**	**DeepRP**	**DeepGRP**
Paint Control	0.032	0.045	0.067	0.052	0.039
IOF/ROL	0.241	0.312	0.425	0.356	0.274
GSDTSR	1.205	1.462	1.934	1.712	1.295

Table 4. depicts the inference time of existing and proposed models on all datasets. In this analysis, the proposed model provides lower inference time compared to TCP-Net++, Actgraph, DeepRP models. However, slightly higher inference time than LogTCP. This difference arises because LogTCP performs minimal log pre-analysis with almost no complex feature extraction, making it extremely fast but also shallow in analysis. DeepGRP, on the other hand, incurs a modest computational overhead due to graph construction and message passing, but this cost enables it to capture higher-order dependencies among neurons. Importantly, the inference time of DeepGRP remains within real-time CI constraints, and its significant gains in APFD, NAPFD, and RMSE far outweigh this marginal overhead. Thus, DeepGRP provides an effective balance between fault detection capability and runtime efficiency for practical deployment in CI environments.

5. Conclusion

This framework suggests using DeepGRP to enhance TC characteristics for unlabelled training TCs for efficient TCP in RT. The DG can extract graph-level neural dynamic features by seeing DNN neurons as nodes and the proximity matrix as the association link among nodes. TCP may benefit from the consolidated node attributes including structural details and adjacent node data. By converting model activations into spatial associations of neurons, the DG solves restrictions depending on model stimulation techniques. Learning single cases assist to prioritize many TCs, so it is more generic than model activation-based approaches. Experimental results show DeepGRP can enhance TCP for large size test suits compared to other standard algorithms.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] Naik, K., and Tripathy, P., *Software Testing and Quality Assurance: Theory and Practice*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011.

[2] Tomar, V., Bansal, M., and Singh, P., "Regression testing approaches, tools, and applications in various environments," in *Proc. 4th Int. Conf. Artif. Intell. Speech Technol. (AIST)*, 2022, pp. 1–6.

[3] Shahin, M., Babar, M. A., and Zhu, L., "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[4] Zampetti, F., Vassallo, C., Panichella, S., Canfora, G., Gall, H., and Di Penta, M., "An empirical characterization of bad practices in continuous integration," *Empirical Softw. Eng.*, vol. 25, no. 2, pp. 1095–1135, 2020.

[5] Debbiche, A., Dienér, M., and Berntsson Svensson, R., "Challenges when adopting continuous integration: A case study," in *Proc. Product-Focused Softw. Process Improve., PROFES 2014*, Helsinki, Finland, Dec. 2014, pp. 17–32.

[6] Yoo, S., and Harman, M., "Regression testing minimization, selection and prioritization: A survey," *Softw. Test., Verif. Rel.*, vol. 22, no. 2, pp. 67–120, 2012.

[7] Sheikh, R., Babar, M. I., Butt, R., Abdelmaboud, A., and Eisa, T. A. E., "An optimized test case minimization technique using genetic algorithm for regression testing," *CMC-Comput. Mater. Contin.*, vol. 74, no. 3, pp. 6789–6806, 2023.

[8] Kazmi, R., Jawawi, D. N., Mohamad, R., and Ghani, I., "Effective regression test case selection: A systematic literature review," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–32, 2017.

[9] M. Alzahrani, A. A. Alshahrani, and M. A. Alhassan, "Test case prioritization techniques in software testing: A systematic review," *J. Softw.: Evol. Process*, vol. 34, no. 7, p. e2330, 2022.

[10] Alkawaz, M. H., and Silvarajoo, A., "A survey on test case prioritization and optimization techniques in software regression testing," in *Proc. IEEE 7th Conf. Syst., Process Control (ICSPC)*, 2019, pp. 59–64.

[11] Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., and Tumeng, R., "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, 2018.

[12] Lima, J. A. P., and Vergilio, S. R., "Test case prioritization in continuous integration environments: A systematic mapping study," *Inf. Softw. Technol.*, vol. 121, p. 106268, 2020.

[13] Wang, Z., Fang, C., Chen, L., and Zhang, Z., "A revisit of metrics for test case prioritization problems," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 30, no. 08, pp. 1139–1167, 2020.

[14] Khaleel, S. I., and Anan, R., "A review paper: Optimal test cases for regression testing using artificial intelligent techniques," *Int. J. Elect. Comput. Eng. (IJECE)*, vol. 13, no. 2, pp. 1803–1816, 2023.

[15] Wei, Z., Wang, H., Ashraf, I., and Chan, W. K., "Predictive mutation analysis of test case prioritization for deep neural networks," in *Proc. IEEE 22nd Int. Conf. Softw. Qual., Rel. Security (QRS)*, 2022, pp. 682–693.

[16] Yan, R., Chen, Y., Gao, H., and Yan, J., "Test case prioritization with neuron valuation based pattern," *Sci. Comput. Program*, vol. 215, p. 102761, 2022.

[17] Han, Y., Chen, G., and Han, B., "An improved method for test case prioritization in continuous integration based on reinforcement learning," in *Proc. 3rd Int. Conf. Manage. Sci. Softw. Eng. (ICMSSE)*, 2023, pp. 958–972.

[18] Rosenbauer, L., Stein, A., Maier, R., Pätzel, D., and Hähner, J., "Xcs as a reinforcement learning approach to automatic test case prioritization," in *Proc. Genet. Evol. Comput. Conf. Companion*, 2020, pp. 1798–1806.

[19] Bagherzadeh, M., Kahani, N., and Briand, L., "Reinforcement learning for test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 2836–2856, 2022.

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

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

[22] Pan, Z., Zhou, S., Wang, J., Wang, J., Jia, J., and Feng, Y., "Test case prioritization for deep neural networks," in *Proc. 9th Int. Conf. Dependable Syst. Their Appl. (DSA)*, 2022, pp. 624–628.

[23] Yaraghi, A. S., Bagherzadeh, M., Kahani, N., and Briand, L., "Scalable and accurate test case prioritization in continuous integration contexts," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 1–27, 2023.

[24] Chen, Z. et al., "Exploring better black-box test case prioritization via log analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, pp. 1–33, 2022.

[25] Waqar, M., Imran, Zaman, M. A., Muzammal, M., and Kim, J., "Test suite prioritization based on optimization approach using reinforcement learning," *Appl. Sci.*, vol. 12, no. 13, p. 6772, 2022.

[26] Abdelkarim, M., and ElAdawi, R., "TCP-Net++: Test case prioritization using end-to-end deep neural networks-deployment analysis and enhancements," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*, 2023, pp. 99–106.

[27] Chen, J., Ge, J., and Zheng, H., "Actgraph: Prioritization of test cases based on deep neural network activation graph," *Autom. Softw. Eng.*, vol. 30, no. 2, p. 28, 2023.

[28] Manikkannan, D., and Babu, S., "Test case prioritization via embedded autoencoder model for software quality assurance," *IETE J. Res.*, pp. 1–11, 2023.

[29] Spieker, H., Gotlieb, A., Marijan, D., and Mossige, M., "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2017, pp. 12–22.

[30] Elbaum, S., Mclaughlin, A., and Penix, J., "The Google dataset of testing results," 2014.