International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

An Explainable Deep Learning Model in Improving Test Case Prioritization for Continuous Integration Testing

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

[1]Research Scholar, Department of Computer Science, Sri Krishna Adithya College of Arts and Science,
Assistant Professor in Department of Computer Technology and Data Science, Sri Krishna Arts and Science College, Coimbatore, Tamil Nadu, India
Email: *shankar.ramakrishnanphd17[at]gmail.com*

[2]Associate Professor, Department of Computer Science, Sri Krishna Adithya College of Arts and Science, Coimbatore, Tamil Nadu, India
Email: *ekgirisan[at]gmail.com*

Abstract: ***Continuous Integration (CI) testing is a critical phase in current software industry. Test Case Prioritization (TCP) methods are introduced to enhance Regression Testing (RT) by ranking test cases for early developer feedback. Various Deep Learning (DL) models have been developed to improve TCP in CI environments. But, many struggle to simultaneously capture the structural relationships among test case features and the temporal dependencies across multiple testing cycles. Furthermore, these models often rely on large amounts of historical execution data, limiting their effectiveness in fast-paced and diverse CI scenarios. To address this issue, XCG-TCP, an eXplainable Convolutional Neural Network (CNN) – Gated Rectified Unit (GRU) is proposed for accurate TCP in CI. Initially, the collected test case data will be pre-processed using data cleaning, categorical encoding, and feature scaling to ensure balanced and consistent inputs. A Deep CNN (DCNN) extracts spatial and structural features from various test case attributes such as execution duration, previous results, status changes, execution flags, priority and code modification distance. These features are then fed into a Gated Recurrent Unit (GRU) to model the temporal dependencies and sequence patterns across regression cycles enabling effective identification of failing test cases. The integration of SHapley Additive exPlanations (SHAP) as an Explainable Artificial Intelligence (XAI) model enhances the CNN-GRU model by quantifying the influence of each input feature on the final TCP decision. This combination of these models enhances early fault detection, accelerates testing cycles and improves adaptability across varying CI environments. Experimental results demonstrate that XCG-TCP outperforms standard algorithms on industrial datasets.***

Keywords: Continuous Integration, Test Case Prioritization, Convolutional Neural Network, Gated Rectified Unit, SHapley Additive exPlanations

1. Introduction

Software Testing (ST) is essential for software development, intended for identifying bugs and guaranteeing the system functions as expected [1]. Within ST, RT plays a key role in preventing different errors and ensuring that code variations do not introduce issues [2]. Modern projects often adopt CI, a methodology for creating and testing software that streamlines the development phase, including frequent execution of RT [3]. However, running all test cases in every cycle is challenging due to time, resource, and cost constraints, especially with rapid release schedules [4]. RT becomes particularly demanding in large software systems, requiring substantial computational resources and ongoing maintenance [5]. Common RT approaches include minimization, which removes redundant test cases [6]; selection, which chooses the most critical ones [7]; and TCP, which reorders test suites to achieve goals such as early fault detection [8]. TCP improves RT efficiency and quality enabling parallel debugging and testing to reduce overall costs.

TCP can run continuously until resources are depleted or all tests are executed [9]. TCP techniques are broadly classified into code-based approaches, which use Model-driven approaches which rely on system behavior models and codes for determining processing order [10, 11]. Both aim to maximize fault detection within limited resources. DL–based TCP leverages neural networks to automatically learn complex patterns from historical test execution data, code variations and fault trends, effectively handling large test suites [12]. Also, DL models adaptively prioritizes the test cases without manual feature engineering for higher fault detection and better generalization in diverse CI environments [13]. Various DL based TCP models have been developed for CI.

A DL approach for TCP called AnoLSTM [14] for CI was created. In this model, Automatic pattern learning and successful TCP were achieved by utilizing historical test case data, which included features like test case duration, execution, and a list of past test results. Improving the rate of bug discovery in test cases was the primary goal of this strategy. A DL framework was developed [15] to optimize TCP and Test Case Generation (TCG) in ST. Unified Modeling Language (UML) diagrams from historical source code are changed to CSV, features extracted via Entropy-based Locust Swarm Optimization Algorithm (Ent-LSOA) and reduced using Pearson Correlation Coefficient–Generalized Discriminant Analysis (PCC-GDA). Finally, optimal test cases are prioritized for CI using an Interpolated Multiple Time Scale Recurrent Neural Network (IMTRNN). A Gated Recurrent Unit (GRU)-based DL model [16] was presented for TCP in CI testing. By analyzing historical test

execution data and factors such as distance, duration, status changes, and last run, the model ranks crucial test cases for earlier fault recognition and reduced testing time. This approach accelerates feedback in CI workflows enabling rapid defect resolution and enhancing development efficiency.

While several existing models for DL based TCP have demonstrated good performance, they also exhibit notable limitations. In RT, many DL models struggle to capture both the structural relationships within test case features and the temporal dependencies that arise across multiple testing cycles leading to suboptimal prioritization. Additionally, some models often rely heavily on large volumes of past test execution data, that might not necessarily be available in fast-paced CI environments limiting their adaptability to diverse software projects and testing scenarios.

To address these challenges, a new DL–based TCP model called XCG-TCP is developed for accurate TCP in CI environments. The test case data undergoes a pre-processing stage involving data cleaning, categorical encoding and feature scaling to ensure consistent and balanced input. Then, the pre-processed data is inputted to a DCNN which extracts spatial and structural features from test case attributes such as execution duration, last run result, change in status, execution status flags, calculated priority and distance from modified test codes. These feature representations capture the key static relationships within the test case data. The extracted features are then passed to the GRU which models the temporal dependencies and sequence patterns across multiple regression cycles. This temporal learning enables the GRU to identify failing test cases more effectively, allowing for better prioritization decisions. To enhance trust and interpretability, the SHAP model is integrated as an XAI method. SHAP provides transparent insights into the impact of every input attribute on the forecasting made by both the CNN and GRU components for the final TCP actions. By combining deep feature extraction, temporal modeling, and explainability, XCG delivers early fault detection, faster testing cycles, and greater adaptability to diverse CI settings.

The following is the outline of the study: Section II examines associated studies on TCP prediction using DL techniques. Section III presents the proposed algorithm. Section IV evaluates the effectiveness of the suggested approach in relation to current algorithms. Section V completes the study and discusses potential future improvements.

2. Literature Survey

Some of the recent DL based TCP Prioritization models are given below.

Rawat et al. [17] devised a Reinforcement Learning and hidden Markov model for prioritizing software test cases. This model optimizes the selection of test cases that prioritize identifying faults in new code variations presented into the code-bases. Also, it allows the developers to prevent multiple builds with early build failure detection potentially reducing the cost of CI through automated build-result prediction. However, the application of pre-collected datasets prevented the findings from being statistically validated. Abdelkarim & ElAdawi, [18] constructed a TCP with End-to-End Deep Neural Networks (TCP-Net++). This model learns comprehensive connection among test cases and the code itself by integrating characteristics such as test case data, utilization data, and error history. 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. [19] developed a DNN model to prioritize the teat case using the activation graph (Actgraph). In this approach, the DNN neurons are seen as nodes, and the adjacency matrix as the network of connections between them. Then, through message passing, the adjacency matrix and node characteristics were amassed. The resulting node features will include the neighboring nodes' features and the structural information between the nodes, which may be utilized efficiently for TCP. But, this model results with high time complexity issues. da Roza et al. [20] created a contextual information-based approach for machine learning-based TCP in CI development. The model prioritizes test cases with higher failure probability and other properties, such as execution time, size and complexity. It uses two contextual algorithms like Multi-Armed Bandit (MAB) and Random Forest (RF) to evaluate feature groups simply composed during the CI cycle. However, delayed learning and slow convergence occurred due to ineffective reward levels in CI iterations.

Manikkannan & Babu, [21] suggested a TCP using Embedded Auto Encoder (EAE) for Software Quality Assurance and to generate a well-organized system of the prioritized test cases. The code analysis for every benchmark was initially processed from the source code to reduce noise. 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. Garg & Shekhar, [22] suggested a ranking-based Non-Dominated Sorting Genetic Algorithm (NSGA-2) algorithm for TCP optimization to enhance the fault sensitivity analysis. This model prioritizes test cases delicate to detects which was particularly produced by new alterations. Important goals including sensitivity index, execution cost, and average percentage of faults detected (APFD) are achieved by utilizing previous data. This method was tested on eight applications including five handcrafted and java-based applications for fault sensitivity analysis. But, this model necessitated more features to increase scalability and efficiency.

Silveira et al. [23] suggested a visual analytics system for exploring TCP called TPVis. In this method, TPVis was developed in collaboration with ST professionals and features 12 analytical tools to support visualization and prioritization. The model effectively addresses two use cases and incorporates feedback from domain experts. Additionally, it supports generic datasets, allowing it to be used across different software projects and integrated with CI pipelines. But, very little information about a test case and its execution history were used which limits the models

performances. Assiri [24] presented a TCP method using the Dragon Boat Optimization Algorithm (DBOA) for software quality testing. Inspired by dragon boat racing, this model reorders test cases to enhance fault detection and reduce execution time, optimizing APFD for faster coverage. DBOA excels at handling large search spaces which balances the exploration and exploitation and adapting to complex testing scenarios. However, this model results in higher computational complexity and overfitting issues.

3. Proposed Methodology

In this section, the proposed model is completely illustrated. Figure 1 depicts pipeline of the suggested model.

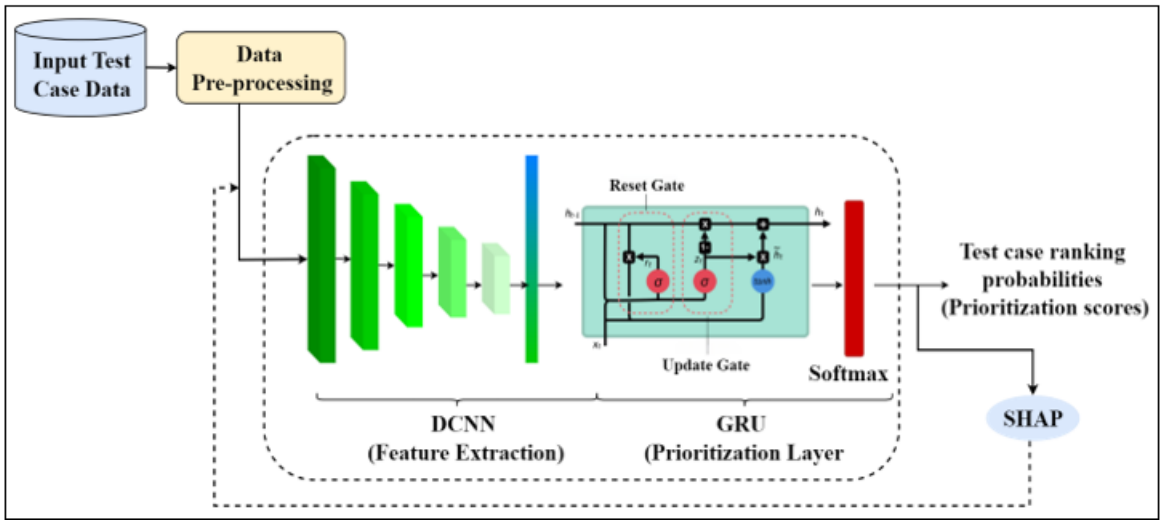

Figure 1: Outline of the suggested work

3.1 Data Pre-processing

In TCP for CI, data pre-processing equips the test case execution data for effective modeling as illustrated below:

- **Data Cleaning:** Removes incomplete or inconsistent test execution records, such as missing execution durations or status flags. Any test cases with corrupted or incomplete historical execution data are either imputed using median values or excluded to maintain dataset integrity.
- **Categorical Encoding:** Converts categorical features like last run result (pass/fail), execution status flags, and change status into numerical formats using One-Hot Encoding. This avoids misleading ordinal interpretations of test case states and preserves categorical distinctiveness.
- **Feature Scaling:** Numerical features such as execution duration, calculated priority, and code modification distance are normalized using Min-Max Scaling to a [0,1] range, ensuring all features contribute proportionally during CNN-GRU training.

These steps ensure the TCP model receives consistent and noise-free inputs reflective of the test case execution context.

3.2 DCNN-GRU

The CNN extracts the spatial and structural relationships from test case attributes at each regression cycle. Three primary components make up the model design: the input layer, the feature extraction layer, and the prioritizing layer. A DCNN at the feature extraction layer processes the pre-processed test case attributes, which are received as input by the input layer. The feature extraction layer's CNN uses a combination of pooling and convolutional layers. Utilizing filters and nonlinear activation functions, the convolutional layers aim to uncover local patterns and structural links from the test case information. The spatial patterns and connections between features that are important for finding test cases with faults are captured by these layers. In order to derive attributes x_{l-1} from the input, the convolutional layer uses a collection of weighted kernels W_l for each layer l. Eq. (1) provides a numerical illustration of this process:

$$C^l = W^{l^T} X^{l-1} + b^l \quad (1)$$

The input is convolved with the kernels and the bias term b^l is added to determine the resultant feature map, which is denoted as C^l. A total of three convolutional layers are utilized: the first utilizes 64 filters with 1 × 5 kernels, the second employs 128 filters with 1 × 3 kernels, and the third employs 256 filters with 1 × 3 kernels. The last convolutional layer contains 128 filters with 1 × 3 kernels, while the fourth layer employs 256 filters with 1 × 3 kernels. To guarantee that the test case attribute sequences may be used for fine-grained feature extraction, every convolutional layer employs a stride of 1 × 1. The 'tanh' function and the 'he_normal' kernel initializer are used by the convolutional layers.

After the convolutional layers, the feature maps are downsampled in the pooling layers that follow. In order to reduce the data's spatial dimensions, this downsampling keeps the most significant features. In order to reduce computational complexity and zero in on the most essential information, the pooling layers reduce the spatial resolution. The full feature map is obtained by doing the pooling technique with a window size (m, n). To get the pooled version, we use the "max" function to pick the highest number in each window. The formula to calculate P^l is the same as in Eq. (2).

$$P^l = Max_{(m,n)} C^l \quad (2)$$

A GRU layer is employed for TCP in the last phase of the XCG-TCP model. The DCNN begins by flattening the spatial and structural characteristics it has retrieved into 1D vectors. Afterwards, the feature vectors are sent to the 1024-unit GRU layer. The GRU processes the sequence of features across multiple regression cycles, capturing temporal dependencies and trends in test case execution patterns. The GRU architecture includes two key elements: the update gate (z_t) and reset gate (r_t), which regulate the flow of information and enable the model to retain or discard historical context as needed. This mechanism allows XCG-TCP to effectively identify and prioritize fault-prone test cases based on both current and past execution data. The computation of GRU layers can be represented in below equations

The update gate z_t is computed as in Eq. (3)

$$z_t = \sigma(W_z(h_{t-1}, x_t) + b_z) \quad (3)$$

The reset gate r_t is computed as in Eq. (4)

$$r_t = \sigma(W_r(h_{t-1}, x_t) + b_r) \quad (4)$$

The hidden state $\hat{h}_t$ is then calculated as in Eq. (5)

$$\hat{h}_t = \tanh(W_h.[r_t \odot h_{t+1}, x_t] + b_h) \quad (5)$$

Finally, the hidden state h_t representing the temporal embedding is updated by

$$h_t = (1 - z_t) \odot h_{t+1} + z_t \odot \hat{h}_t \quad (6)$$

In Eq. (3)-(6), x_t and h_t represent the input feature vector and the hidden state at regression cycle t, respectively. W_z, W_r, W_h corresponds to the weights for the update, reset gates and hidden state, while b_z, b_r, b_h are their respective bias vectors. σ and $\odot$ are the activation function and element-wise multiplication respectively.

Furthermore, the use of the DCNN–GRU architecture in XCG-TCP effectively fuses the merits of DCNN for spatial feature retrieval and GRU for temporal sequence modeling, thereby reducing the impact of noisy or redundant features on prioritization decisions. The architecture is designed with appropriate filters and layers to capture critical feature interactions from test case execution data and reducing background noise for enhanced efficiency.

Next, the above layer's output is distributed over a prioritization scoring layer with a softmax function. This output creates prioritization likelihoods of each test case distinguishing an error. These probabilities are used to generate the final ranking, ensuring that test cases with the highest predicted fault detection potential are initially performed in CI.

3.3 SHapley Additive exPlanations (SHAP)

SHAP is an XAI technique grounded in cooperative game theory designed to quantify the influence of every input feature to models estimation. In XCG-TCP, SHAP is used to interpret the combined CNN-GRU structure by assigning an important value to each test case attributes.

It uses well-formed values to interpret the influence of every features to the forecasting. In the XCG-TCP model, SHAP applies cooperative game theory to evaluate how effectively each subset (or coalition) of input features contributes to improving the TCP outcome. The SHAP formula is defined as in Eq. (7)

$$e(Z') = \varphi_0 + \sum_{j=1}^{M} \varphi_j z'_j \quad (7)$$

Where, z_j denotes the coalition vector i.e., present if $z' = 1$ or absent if $z' = 0$. M denotes input features count, e provides the models explanation. SHAP computes Shapley values by assuming some features are active (present) while others are inactive (absent) which allows SHAP to determine each feature's contribution to the prediction. In order to determine the SHAP values for the function f, where S is a subset of the features and Z is a collection of every feature input with $(z' = 1)$, it is necessary to identify the anticipated outcome of the unit given that S is a subset of the input features and write it as $E[(f(x)|x_s)]$. SHAP values relate to the φ_j values for feature j in game theory is computed by averaging its marginal contribution across all possible feature subsets S that do not include j as in Eq. (7)

$$\varphi_j = \sum_{S \subseteq Z \setminus \{j\}} \frac{|S|!(M-|S|-1)!}{M!} [f_x(S \cup \{j\}) - f_x(S)] \quad (8)$$

Where, $f_x(S)$ will be model output using only S attributes. This formulation ensures a fair distribution of the model's prediction among all features based on their cooperative influence.

For XCG-TCP, SHAP values are computed for the final prioritization score produced by the CNN–GRU network. These values are visualized to highlight which features most strongly influence the ranking of test cases, helping practitioners validate the model's reasoning and build confidence in automated prioritization decisions.

4. Result and Discussion

4.1 Dataset Description

To demonstrate the efficacy of the suggested model, we employ the following resources: the Google Shared Dataset of Test Suite Results (GSDTSR) [26], industrial datasets for testing complicated industrial robots from ABB Robotics Norway, Paint Control [25] and IOF/ROL [25] from Google. With data spanning more than 300 CI cycles, these datasets provide an overview of test runs and their results. All of these are implemented in a CI setting for software development.

Table 1: Illustration of Industrial Data Sets: All columns represent dataset size

Dataset	*Test Cases*	*CI Cycles*	*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%

The structure of the data sets is summarized in Table 1. While GSDTSR is divided into hourly intervals because it initially provided log data of 16 days, which is too short for our review, both ABB data sets are divided into daily intervals. Even still, GSDTSR has fewer collapsed test executions and a larger average test suite size per CI cycle than the ABB data sets. The above mentioned test cases are defined by their execution duration, their previous last execution time and results of their recent executions. Paint-Control, IOFROL and Google GSDTSR are datasets that contain only features related to the execution history of test cases.

4.2 Parameter Settings

The proposed XCG-TCP model is trained using five convolutional layers with 32, 64, 128, 128, and 64 filters respectively, each having a kernel size of 1 × 3 and stride of 1 × 1. All convolutional layers use the tanh function, he_normal kernel initializer, and max pooling with pool size 1 × 2 where applicable. The GRU layer has 1024 units with tanh activation for candidate states and sigmoid for the update and reset gates. The final Softmax layer outputs test case ranking likelihoods. With a learning rate of 0.001, batch size of 64, and 100 epochs, the model is trained using the Adam optimizer with categorical cross-entropy loss.

4.3 Performance Evaluation

The performance of XCG-TCP is compared with the existing algorithms like AnoLSTM [14], IMTRNN [15], GRU-TCP [16], TCP-Net++ [19] DBOA-TCP [24] is executed in Python 3.7.8 using the dataset illustrated in section 4.1. An overview of the measures used to compare the suggested and prevailing approaches is provided below.

4.3.1 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.

$$APFD = 1 - \left(\frac{TF_1 + TF_2 + \cdots + TF_m}{NM} + \frac{1}{2n}\right) \quad (9)$$

In Eq. (9), m is the aggregate amount 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.

4.3.2 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.

$$APFD_c = \frac{\sum_{i=1}^{m}(f_{i*}(\sum_{j=TF_i}^{n} t_j - \frac{1}{2}tTF_i))}{\sum_{j=TF_i}^{n} t_{j*} \sum_{i=1}^{m}(f_i)} \quad (10)$$

In Eq. (10), 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.

4.3.3 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. (11),

$$NAPFD = p - \left(\frac{TF_1 + TF_2 + \cdots + TF_m)}{m*n} + \frac{p}{2n}\right) \quad (11)$$

In Eq. (11), 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.

4.3.4 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. (12),

$$APFD_a = \left(1 - \sum_{i=1}^{m} \frac{\sum_{j=1}^{TF_i} C_j}{m \sum_{j=1}^{n} C_j}\right) * 100\% \quad (12)$$

In above Eq. (12), C_j is the cost obtained in the j^{th} TC.

4.3.5 Time-aware average percent of faults detected ($APFD_{TA}$): This is the specific instance of 〖APFD〗_c for cases where the test costs and mistake challenges are the same.

$$APFD_{TA} = \frac{\sum_{i=1}^{m}(\sum_{j=TF_i}^{n} C_j - \frac{1}{2}CTF_i))}{\sum_{j=1}^{n} C_{j*}|\sigma|} \quad (13)$$

In Eq. (13), σ denotes constant among the primary and latter TCs in the whole TS.

4.3.6 Root mean square error (RMSE): It determines the difference amongst the observed and estimated NAPFD values. Using the dissimilar value intended for T' in a CI repetition specified by learning model ($\hat{u}_C$), the adjoint estimated value T' is used by RL model.

$$RMSE\ (\Psi) = \sqrt{\frac{\sum_{q}^{CI}(\hat{u}_q - u_q)}{CI}} \quad (14)$$

In Eq. (14), 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.

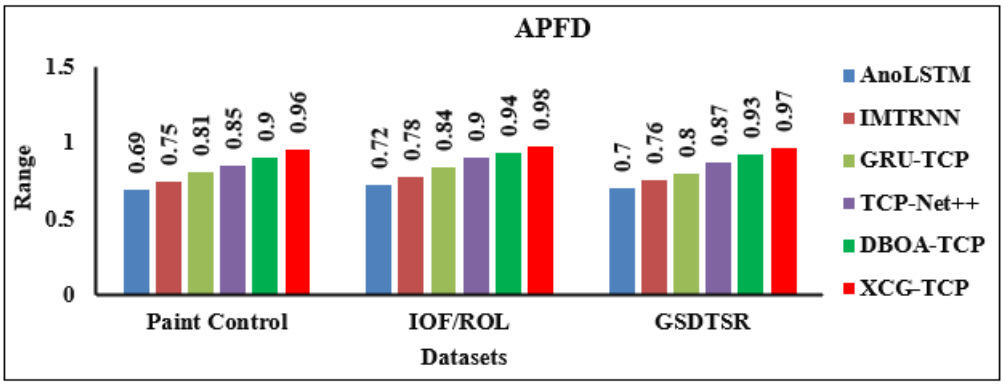

Figure 2: APFD Evaluation for Proposed and Existing Models

Figure 2 demonstrates the APFD analysis for suggested and existing models on different test cases dataset. In figure 3, XCG-TCP model clearly show more efficiency than conventional models for TCP on collected industrial dataset. For instance, APFD value of XCG-TCP is 39.13%, 28%, 18.52%, 12.94% and 6.67% higher than the AnoLSTM, IMTRNN, GRU-TCP, TCP-Net++ and DBOA-TCP on Paint Control dataset. Likewise, Figure 3 shows that the suggested model attained higher $APFD_c$ than other methods. For example, $APFD_c$ value of XCG-TCP is 44.78%, 31.08%, 22.78%, 11.49% and 3.19% greater than the AnoLSTM, IMTRNN, GRU-TCP, TCP-Net++ and DBOA-TCP models respectively on IOF\ROL dataset.

The figure 4 establishes the assessment of NAPFD values of suggested and traditional models on various test case dataset like Paint Control, IOF/ROL and GSDTSR. For instances, the NAPFD value of XCG-TCP is 24.68%, 17.07%, 12.94%, 9.09% and 4.35% higher than the AnoLSTM, IMTRNN, GRU-TCP, TCP-Net++ and DBOA-TCP on GSDTSR dataset. Similarly, the figure 5 establishes the assessment of $APFD_a$ values of suggested and traditional models on various test case dataset. The $APFD_a$ value of XCG-TCP is 26.32%, 18.52%, 12.94%, 7.87% and 3.23% higher than the AnoLSTM, IMTRNN, GRU-TCP, TCP-Net++ and DBOA-TCP on IOF/ROL dataset. In both the comparison, the proposed XCG-TCP models achieves high NAPFD and $APFD_a$ results than other existing models.

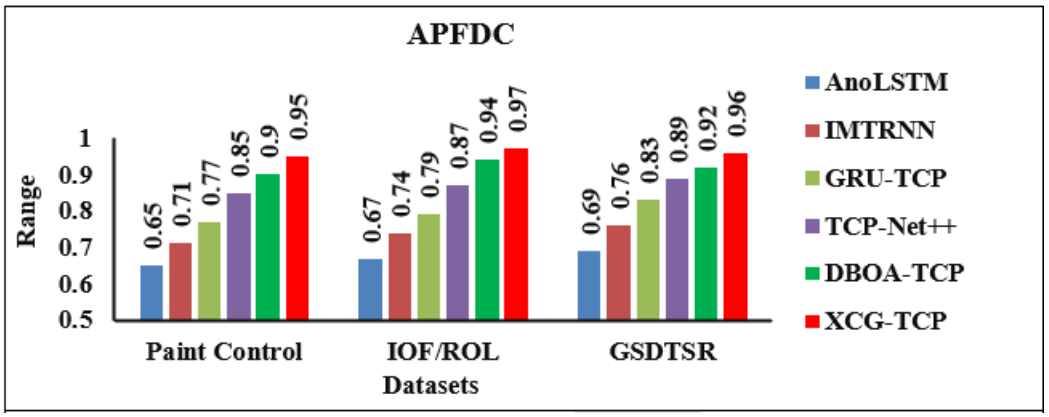

Figure 3: $APFD_c$ Analysis for various methods

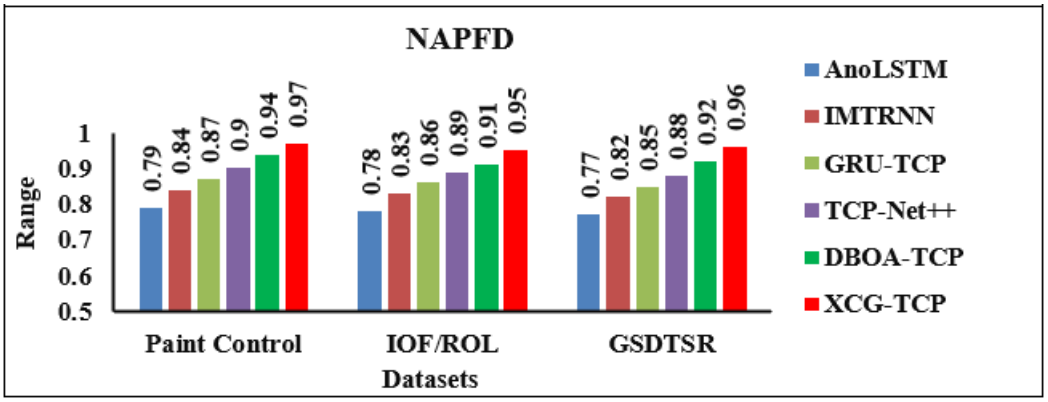

Figure 4: NAPFD Evaluation for Proposed and Existing Models

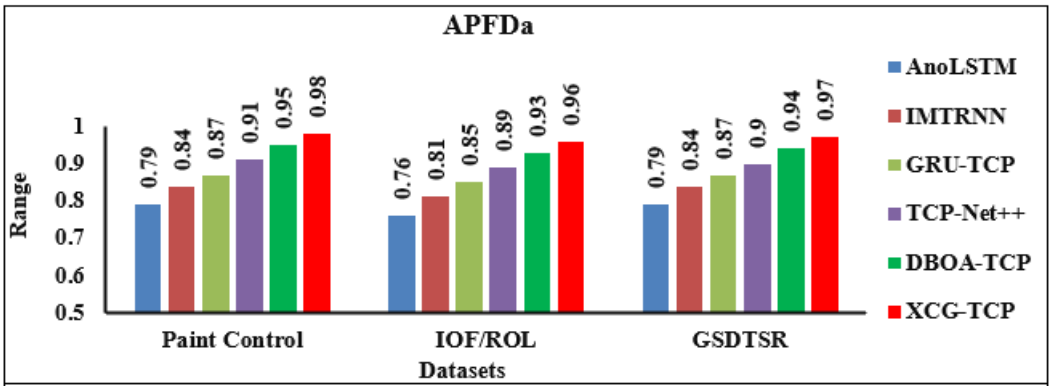

Figure 5: $APFD_a$ Analysis for various methods

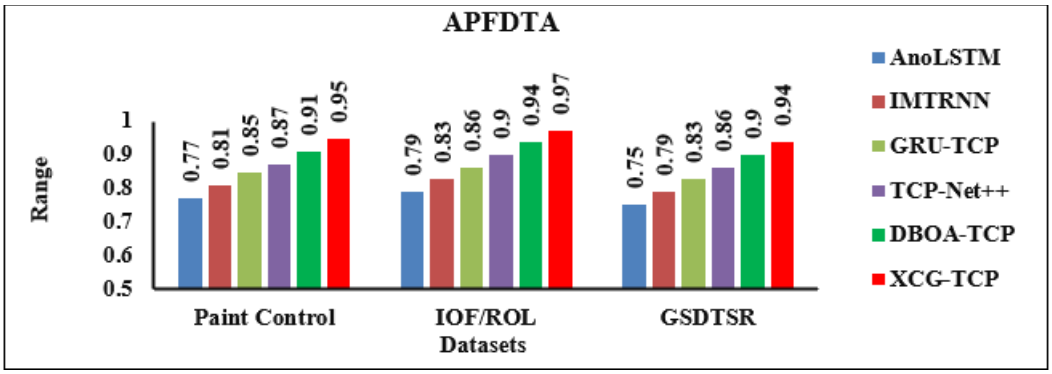

Figure 6: $APFD_{TA}$ Analysis for various methods

The figure 6 establishes the assessment of $APFD_{TA}$ values of suggested and traditional models on various test case dataset like Paint Control, IOF/ROL and GSDTSR. In figure 7, XCG-TCP attains superior $APFD_{TA}$ than earlier TCP algorithms. The $APFD_{TA}$ value of XCG-TCP is 25.33%, 18.99%, 13.25%, 9.3% and 4.44% higher than the AnoLSTM, IMTRNN, GRU-TCP, TCP-Net++ and DBOA-TCP algorithms on GSDTSR dataset. The figure 7 establishes the assessment of RMSE values of suggested and traditional models on various test case dataset. The RMSE value of XCG-TCP is 75.73%, 72.56%, 58.49%, 50.35% and 33.69% higher than the AnoLSTM, IMTRNN, GRU-TCP, TCP-Net++ and DBOA-TCP algorithms on Paint Control dataset. This study shown that, in comparison to other classical models, the suggested model has lower RMSE values.

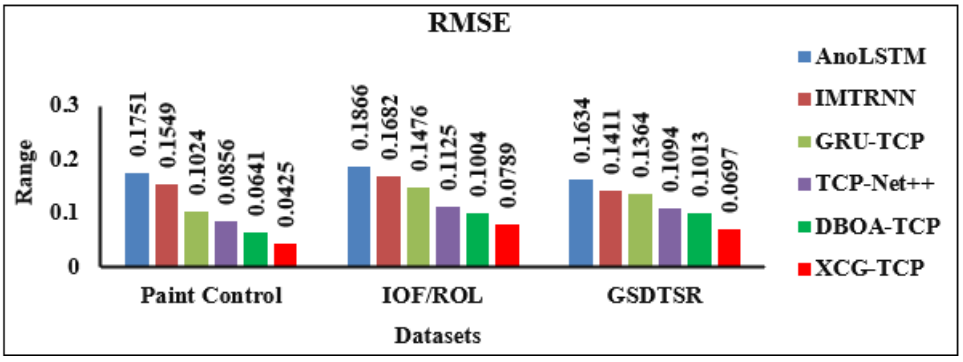

Figure 7: RSME Analysis for various methods

5. Conclusion

In this paper, XCG-TCP model is developed integrating CNN, GRU and SHAP for accurate TCP in CI. Test case data undergoes cleaning, encoding, and scaling for balanced inputs. A Deep CNN extracts spatial and structural features from attributes like execution time, results history, status changes, flags, priority, and code change distance. These features are processed by a GRU to capture temporal patterns across regression cycles for detecting failing cases. SHAP, as an Explainable AI method, quantifies each feature's impact on prioritization. This integration improves early fault detection, speeds testing, and adapts to diverse CI settings. Experimental results demonstrate that XCG-TCP outperforms standard algorithms on industrial datasets.

References

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

[2] M. Qasim, A. Bibi, S. J. Hussain, N. Z. Jhanjhi, M. Humayun, and N. U. Sama, "Test case prioritization techniques in software regression testing: An

overview," International Journal of Advanced and Applied Sciences, 8(5), pp. 107-121, 2021.
[3] E. Soares, G. Sizilio, J. Santos, D. A. da Costa, and U. Kulesza, "The effects of continuous integration on software development: a systematic literature review," Empirical Software Engineering, 27(3), p. 78, 2022.
[4] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M. A. Storey, "Uncovering the benefits and challenges of continuous integration practices," IEEE Transactions on Software Engineering, 48(7), pp. 2570-2583, 2021.
[5] D. Elsner, F. Hauer, A. Pretschner, and S. Reimer, "Empirically evaluating readily available information for regression test optimization in continuous integration," in Proc. 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 491-504, Jul. 2021.
[6] S. Parida, D. Rath, and D. B. Mishra, "A review on test case selection, prioritization and minimization in regression testing," in Proc. International Conference on Metaheuristics in Software Engineering and its Application, Cham: Springer International Publishing, pp. 156-163, Feb. 2022.
[7] 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, 27(2), p. 29, 2022.
[8] A. Samad, H. Mahdin, R. Kazmi, and R. Ibrahim, "Regression test case prioritization: a systematic literature review," International Journal of Advanced Computer Science and Applications, 12(2), 2021.
[9] 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, 33(9), pp. 1041-1054, 2021.
[10] N. Gokilavani and B. Bharathi, "Test case prioritization to examine software for fault detection using PCA extraction and K-means clustering with ranking," Soft Computing – A Fusion of Foundations, Methodologies & Applications, 25(7), 2021.
[11] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella, "Single and multi-objective test cases prioritization for self-driving cars in virtual environments," ACM Transactions on Software Engineering and Methodology, 32(2), pp. 1-30, 2023.
[12] A. Sharif, D. Marijan, and M. Liaaen, "Deeporder: Deep learning for test case prioritization in continuous integration testing," in Proc. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 525-534, Sept. 2021.
[13] 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 Proc. 29th International Symposium on Software Testing and Analysis, pp. 177-188, 2020.
[14] A. Tamizharasi and P. Ezhumalai, "A novel framework for optimal test case generation and prioritization using Ent-LSOA and IMTRNN techniques," Journal of Electronic Testing, pp. 1-24, 2024.
[15] T. K. Choudhury, M. Behera, S. K. Dash, S. K. Pani, and J. Mishra, "AnoLSTM – A deep learning approach for test cases prioritization," Procedia Computer Science, 258, pp. 1793-1803, 2025.
[16] A. Behera and A. A. Acharya, "An effective GRU-based deep learning method for test case prioritization in continuous integration testing," Procedia Computer Science, 258, pp. 4070-4083, 2025.
[17] N. Rawat, V. Somani, and A. K. Tripathi, "Prioritizing software regression testing using reinforcement learning and hidden Markov model," International Journal of Computers and Applications, 45(12), pp. 748-754, 2023.
[18] M. Abdelkarim and R. ElAdawi, "TCP-Net++: Test case prioritization using end-to-end deep neural networks – deployment analysis and enhancements," in Proc. 2023 IEEE International Conference On Artificial Intelligence Testing (AITest), pp. 99-106, Jul. 2023.
[19] J. Chen, J. Ge, and H. Zheng, "Actgraph: Prioritization of test cases based on deep neural network activation graph," Automated Software Engineering, 30(2), p. 28, 2023.
[20] E. A. da Roza, J. A. do Prado Lima, and S. R. Vergilio, "On the use of contextual information for machine learning-based test case prioritization in continuous integration development," Information and Software Technology, 171, p. 107444, 2024.
[21] D. Manikkannan and S. Babu, "Test case prioritization via embedded autoencoder model for software quality assurance," IETE Journal of Research, pp. 1-11, 2024.
[22] K. Garg and S. Shekhar, "Optimizing test case prioritization through ranked NSGA-2 for enhanced fault sensitivity analysis," Innovations in Systems and Software Engineering, pp. 1-22, 2024.
[23] J. A. Silveira, L. Vieira, and N. Ferreira, "TPVis: A visual analytics system for exploring test case prioritization methods," Computers & Graphics, 124, p. 104064, 2024.
[24] M. Assiri, "Test case prioritization using dragon boat optimization for software quality testing," Electronics, 14(8), p. 1524, 2025.
[25] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in Proc. 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 12-22, Jul. 2017.
[26] S. Elbaum, A. Mclaughlin, and J. Penix, "The Google dataset of testing results," 2014.