International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I6P121

Volume 74 Issue 6, 294-308, June 2026


*Original Article*

# Deep-TCP: An Intelligent Deep Learning Model for Automated Test Case Prioritization Systems using Neural Networks

Sowmyadevi S[1], Shashi Mehrotra[2]

[1,2]*Department of Computer Science and Engineering, SRM Institute of Science and Technology, Delhi NCR campus, Modinagar, Ghaziabad, Uttar Pradesh, India.*

[1]*Corresponding Author : ss2860@srmist.edu.in*

Received: 05 January 2026 Revised: 30 March 2026 Accepted: 20 April 2026 Published: 27 June 2026

***Abstract*** *- Test case prioritization is the process of ordering test cases based on their importance to improve regression testing efficiency. Test case prioritization approaches have been demonstrated to improve regression testing processes. But operating the whole regression test suite can be inconvenient and costly, particularly for large systems. To overcome this problem, a novel An Intelligent Deep Learning Model for Automated Test Case Prioritization Systems using Neural Networks (Deep-TCP) has been proposed. Test cases are collected from the source code repository and split into test steps. The test steps are preprocessed using Normalization, Tokenization and stemming to remove noise. After pre-processing, word embedding is computed and the embedded features are clustered using K-means clustering. The novelty of the proposed integration is where the effort is of word embedding, K-means clustering, Radial Basis Function Network (RBFN), and CNN-BiGRU to capture both semantic and sequential connections between test cases. Radial Basis Function Network (RBFN) is used for extracting the relevant features and Convolutional Neural Network-Bidirectional Gated Recurrent Unit (CNN-BiGRU) is used for testcase priority such as high priority, average priority and low priority. The proposed framework has been implemented using Python (PyTorch) and evaluated on a system with NVIDIA RTX 4090 GPU. The efficacy of the proposed Deep-TCP framework has been determined using evaluation metrics such as Average Percentage of Faults Detected (APFD). Average APFD of the proposed method is 94.5% which is higher than 71.25% at ATRL-TCP, 73.5% at QAOA-TCS and 81.5% at BootQA approaches.*

***Keywords*** *- Convolutional Neural Network-Bidirectional Gated Recurrent Unit, Fault Detection, K-means clustering, Software Testing, Testcase Prioritization.*

## 1. Introduction

Software testing is structured verification activity used to evaluate and improve software quality [1]. It is also a long-established part of the software lifecycle, evolving alongside software development itself [2]. In practice, testing serves as a quality-assurance mechanism that supports both assessment and attainment of required quality levels [3]. Through systematic execution and evaluation, it helps validate multiple quality attributes-including dependability, usability, security, capability, efficiency, portability, maintainability, and interoperability-by revealing defects and weaknesses in the system [4]. Notably, testing does not solely involve the defect discovery but also aids in defect resolution as well as offering supporting evidence that the system put in place matches the requirements and acceptance criteria [5].

Whenever testing is done thoroughly, defects are likely to be identified at an earlier stage, and correction will be less expensive and more probable of the final product aligning to the needs and expectations of the users [6]. Continuous Integration (CI) is now a popular approach to modern development processes based on automated pipelines, which compile, package, and verify software based on set rules and quality gates [7]. Automation in its turn makes the execution quicker, ensures more repeatability, and less manual work in both the execution of the test and the detection of defects [8]. Nonetheless, the application of the automation frameworks in practice is frequently limited by organizational factors, such as technology shortages, budget issues, and lack of personnel to design, develop, and operate automation resources in the most efficient way possible [9]. As a result, both research and industry have been gravitating toward more automated test design methods, in which language models are becoming more and more used to help in the generation of test cases and other associated testing artifacts [10].

In this scenario, test case prioritization provides a practical mechanism in enhancing good testing outcomes

when limited time or resources budgets are available, by prioritizing test cases in order of their most valuable ones to run first [11]. Test Case Prioritization (TCP) is of great essence when software is in the process of development and maintenance, where regression tests should be repeated and they should be done effectively [12]. Because prioritization necessarily requires making choices about the most important things (e.g., risk, cost, coverage, historical failure trends), the rigorous prioritization of competing characteristics is of central concern to the technique [13]. TCP techniques are designed to mitigate the high execution cost of regression testing by scheduling test cases to maximize early fault detection; therefore, tests with stronger fault-revealing potential and lower execution time are typically placed earlier in the execution order to accelerate feedback and reduce debugging latency [14]. However, as the test suite grows in size, regression testing becomes more time-consuming and resource-intensive [15]. TCP improves fault detection and testing efficiency by prioritizing critical and independent test cases [16]. The main constraint of cost-based test case prioritization is that it requires source code analysis, which makes it difficult to use when source code is unavailable [17]. Although numerous TCP methods are being developed and tested on open-source projects, there is a lack of comprehensive comparative studies on large-scale industrial applications [18].

Despite significant progress in test case prioritization techniques, several challenges remain unresolved. These TCP methods have been found to work in many existing TCP methods that are based on heuristic or search-oriented methods, which lack the semantic relationships and sequential dependencies between steps in test cases. This means accuracy in the prioritization and performance in fault detection can be reduced in the case of using large and complex test suites. As such, it is necessary to have intelligent learning-based systems that can derive meaningful representations on case tests and enhance the effectiveness of prioritization. In order to overcome such constraints, Deep-TCP, a smart deep learning-based test case prioritization framework that uses neural networks has been suggested. The major contribution of the work has been followed by:

- Test cases are collected from software repositories and decomposed into individual test steps to capture detailed testing behavior for prioritization.
- Textual test case data is normalized, tokenized and cleaned using Natural Language Processing (NLP).
- Textual test steps are translated to semantic vectors through word embedding techniques so as to be effectively learned.
- It presents a clustering and feature extraction method based on K-means and Radial Basis Function Network (RBFN) to classify and select test case features of importance.
- To ensure that test cases are detected in the early stages to identify fault early, a hybrid CNN-BiGRU network is created to rank the cases into high, medium, and low priorities.

Section 2 exhibits proposed literature study; Section 3 represents the suggested approach; Section 4 illustrates model outcomes and discussions; and Section 5 indicates the future work's conclusion.

## 2. Literature Review

In 2025, Su, Q., et al [19] proposed Attention Transfer Reinforcement learning for Test Case Prioritization (ATRL-TCP) to dynamically adjust test case prioritization. The point is that the existing model is not able to handle various ranking situations within the framework of CI. The offered ATRL-TCP algorithm dynamically adjusts its input representation based on attention transfer and enables a high level of generalization and enhancement of the performance in ranking test cases in diverse CI settings. The findings show that ATRL-TCP works well in generalization and ranking within a realistic timeframe.

In 2025, Trovato, A., et al [20] proposed a Quantum Approximate Optimization Algorithm (QAOA-TCS) for efficient effective test case selection. Performing the complete set of regression tests may be time-consuming and costly, particularly for large-scale systems. The advantage of the QAOAs outperforms baseline algorithms in terms of efficacy and efficiency, while being equivalent to proposed model. The results show that QAOA-TCS beyond conventional techniques in effectiveness and performs similarly to proposed model in efficiency, demonstrating strong potential.

In 2024, Laaber, C., et al [21] proposed History-Based Greedy Prioritization techniques (HGP) to give micro benchmarks priority in order to speed up detection. Conventional TCP algorithms can be unrealistic with long execution time and poor performance with micro benchmarks. History-based greedy method provides immense efficiency with minimal run time overhead and coverage is less intricate. The result shows that simple, non-coverage-based strategies are more effective in micro benchmarks than the complex coverage-based strategies.

In 2024, Wang, X., et al [22] proposed a Boot Quantum annealing (BootQA) is used to solving test case minimization problem in software. Current quantum annealers are restricted by the number of qubits and cannot provide quantum benefits. QA-based TCM formulation with bootstrap sampling to increase qubit efficiency and solution quality. The BootQA outperforms the other three optimization procedures in terms of time efficiency for addressing big TCM issues. In 2024, Zhang, Z., et al [23] proposed a Density-based Clustering Test Case Prioritization (DC-TCP) is used to prioritize the test case

using DBSCAN-based clustering. Existing diversity-based static black-box prioritizing approaches are time consuming and unstable due to the large computational overhead associated with string distance computations. DC-TCP improves generality through input mapping, improves classification with DBSCAN, and increase sequence merging efficacy with the search techniques. The proposed DC-TCP approach beats static black-box sorting methods in terms of defect detection rate and time economy.

In 2021, Sharif, A., et al [24] proposed a DeepOrder learning algorithm is used to prioritize test cases in continuous integrated. The existing CI test prioritizing algorithms struggle with huge amounts of test data or rely on a small number of pervious cycles, lowering fault detection efficacy. DeepOrder generates effective test ranking by using any amount of test data and learning failure patterns based on criteria like length and execution status. Experimental results reveal that DeepOrder outperforms industry standards and cutting-edge approaches in terms of time efficiency and detect identification.

In 2025, John, A., et al [25] proposed an AI-Driven Test Case optimization for CI (AI-TCO) is used to intelligently select, prioritize, and execute test case. Existing model becomes weak and inefficient as software complexity increases, resulting in duplicate test executions. The AI-driven strategy employs machine learning and predictive analytics to eliminate needless and improve problem identification. The result indicates that AI-optimized testing greatly speeds CI processes while preserving or enhancing overall software quality.

In 2026 Li, Z., et al., [26] proposed environment Adaptation Agent-based RL-TCP method (EAA) that detects significant environmental changes by analyzing fluctuations in prioritization effectiveness. In case of a change being detected, it assigns targeted rewards to test cases. EAA also optimizes the gradient update of the agent in such a way that it incorporates better the dynamics of the environment into retraining. Tests on 12 real-life industrial data and minimizes the. average TTF 35.85 -50.37 positions versus state-of-the-art RL-TCP techniques. In 2023 Singh, A., et al., [27] provided categorization of existing TCP methods that are investigated according to the six research issues that are more pertinent to TCP. The authors used a variety of search terms to retrieve several publications from relevant repositories of journals, conferences, seminars, and symposiums in order to conduct a Systematic Literature Review (SLR) on TCP approaches. This SLR goes into great length on the limitations, possible benefits, and advantages of each TCP technique.

**Table 1. Comparative analysis of existing studies**

| S. No | Author | Approach | Advantage | Limitation |
|---|---|---|---|---|
| 1. | Su, Q., [19] | ATRL-TCP | Adaptive input selection increases ranking accuracy and generalization. | Requires accurate historical data and increases computing overhead. |
| 2. | Trovato, A., [20] | QAOA-TCS | When compared to traditional methods, it improves the efficacy and competitiveness of test case selection. | Requires quantum computing resources, which limits its practical application in existing CI systems. |
| 3. | Laaber, C., [21] | HGP | Prioritizes micro benchmarks quickly and efficiently, with low runtime overhead. | Applicability to complicated systems requiring coverage-based analysis is limited. |
| 4. | Wang, X., [22] | BootQA | Improves the time efficiency and solution quality for large-scale test case minimization with limited qubit resources. | Still limited by existing quantum annealing hardware constraints and scalability. |
| 5. | Zhang, Z., [23] | DC-TCP | Avoids costly string distance computations, which improves defect detection and time efficiency. | Clustering quality depends on DBSCAN parameter choices and data distribution. |
| 6. | Sharif, A., [24] | Deep Order | Handles vast amounts of test data effectively and enhances fault detection while saving time. | The complexity of deep learning models may lead to increased training time and resource consumption. |
| 7. | John, A., [25] | AI-TCO | Reduces duplicate test executions and dramatically accelerates continuous integration while retaining software quality. | Performance is determined on quality of the training data, and system becomes more complicated, significant model tuning may be required. |

Based on the review of the literature, currently, primary methods of test case prioritization are based on heuristic optimization, reinforcement learning, quantum methods, or historic strategies. In spite of the fact that these methods

enhance prioritization and execution efficiency, most of them do not reflect semantic relationships between test case steps and cannot be easily adopted in large and dynamic testing environments. Moreover, a number of them do not have developed deep learning mechanisms to extract structural and serial patterns in data of test cases.

Consequently, the accuracy of priority and the detection of faults is low in the complex software systems. In order to solve these problems, the Deep-TCP architecture introduces the NLP-based preprocessing, clustering, RBFN feature extraction, and the hybrid CNN-BiGRU architecture to achieve the effective automated prioritization of test cases.

## 3. Proposed Methodology

In this section, a DL-based Deep-TCP model is designed to perform effective automated test case prioritization in software testing. The given solution combines several processing steps, such as data preprocessing, word embedding, clustering, feature extraction, and the classification, to increase the quality of test case data and identify meaningful trends in test steps. The overall K-means clustering, Radial Basis Function Network (RBFN), and CNN-BiGRU model allow learning both the semantic and sequential relationship between cases in the test and generates a high, average, and low priority level with accuracy. Figure 1 shows the overall workflow of proposed Deep-TCP methodology.

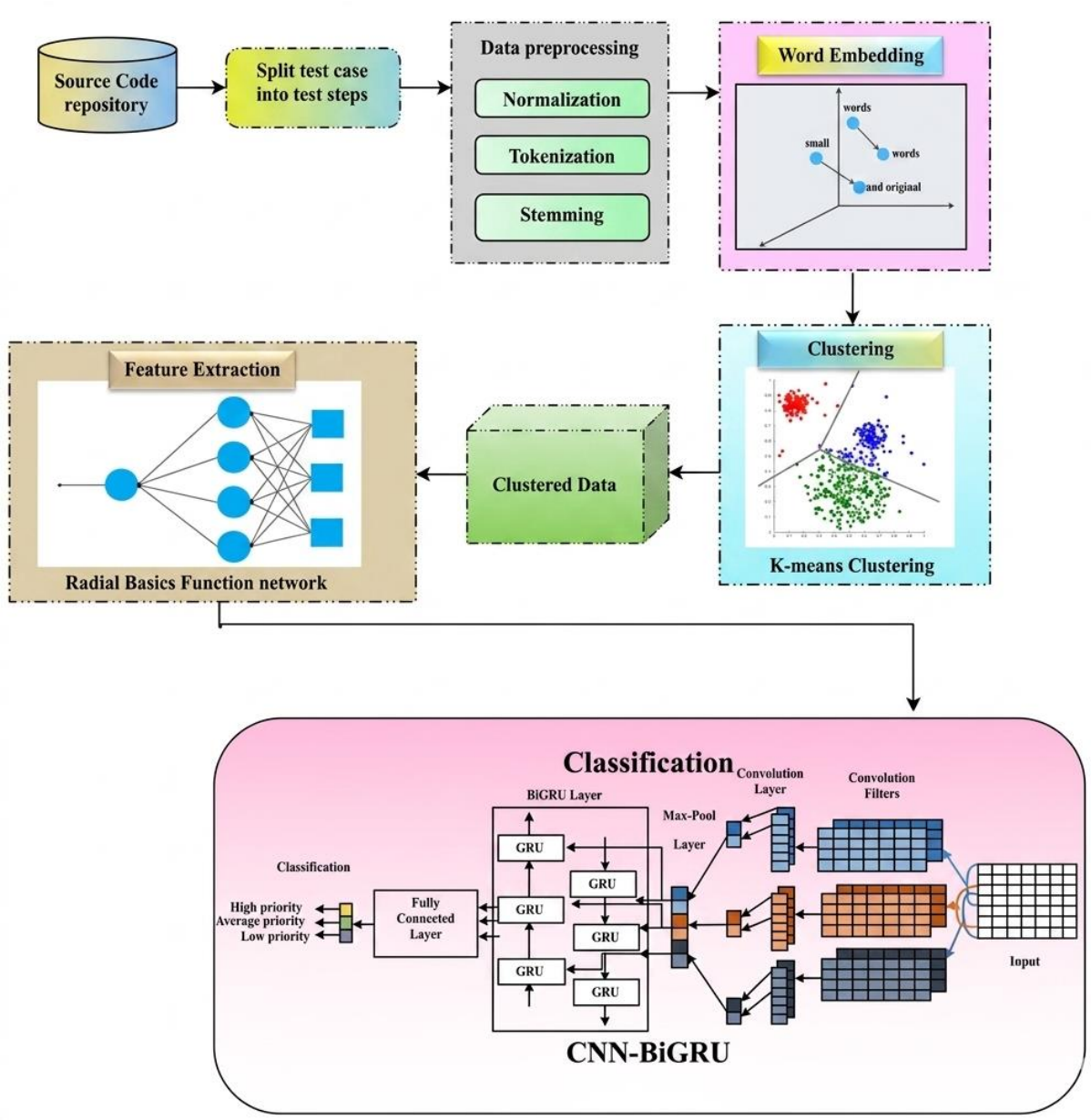


**Fig. 1 Proposed deep-TCP methodology for priority identification**

### *3.1. Data Collection*

The source code repository is used to obtain test cases to verify the accuracy and consistency of the software. These test cases consist of elaborate details like test steps, input conditions as well as anticipated outputs which are utilized to confirm the proper performance of the software system. Effective software testing is enhanced by collecting the test cases properly to aid in the detection of the errors and to make certain reliability and integrity of software. The assembled data is the main input of the suggested Deep-TCP framework that allows to conduct additional preprocessing, extract features, and rank test cases to conduct regression tests that work best.

### *3.2. Data Preprocessing*

Preprocessing of the collected test cases is in form of test steps. There are different operations and anticipated results in each test case. Pre-processing steps are then applied to the test steps to normalize, tokenize and stem the data to be further processed and trained.

#### *3.2.1. Normalization*

Normalization method for preparing data that scales numerical values inside a specific range, typically [0, 1] or [-1, 1]. This reduces the influence of different scales of data and ensures that all characteristics contribute equally to the learning process. Normalization in test case prioritize enhances machine learning accuracy by ensuring consistent and similar data.

Min-Max Normalization converts a feature $Y$ into a scaled version $Y'$ using a formula.

$$Y' = \frac{Y - Y_{min}}{Y_{max} - Y_{min}} \quad (1)$$

Where $Y$are the original values, $Y_{min}$ and $Y_{max}$ are the features minimum and maximum values, and $Y'$ is the normalized value within the range [0,1].

#### *3.2.2. Tokenization*

Tokenization is a data preparation approach that breaks down raw data, such as text or structured test case records, into smaller meaningful pieces called tokens. Tokenization helps simplify test case prioritize and data by breaking down complicated words, and segmenting time-series data for machine learning. This procedure effectively structures and encodes data for analysis with models.

A basic example of word tokenization using a function $U(y)$ may be expressed as

$$U(y) = \{u_1, u_2, u_3, \dots, u_o\} \quad (2)$$

Where, tokenized output is represented by $U(y)$, while individual tokens are denoted by $u_1, u_2, \dots, u_o$.

#### *3.2.3. Stemming*

The streaming technique is a crucial step in Natural Language Processing or text categorization, converting words to their associated root or stem. Stems are a suffix is the combination of a root and its derivation, including prefixes and postfixes. There are two forms of stemming: stem or light stemming and root-based stemming. The former involves eliminating suffixes and prefixes from a word to derive its root. The latter has three sub-categories. There are three types of bases: dictionary-based, no-dictionary, and hybrid.

### *3.3. Word Embedding*

The study employed word2vec, which has been shown to produce Vector representations of words typical NLP tasks than other approaches. Word embeddings can be expressed as a mapping $W \to S^E : x \mapsto \theta$, which converts word $w$ from vocabulary $V$ to real valued vector in an embedding space of dimension proposed the skip-gram architecture, which consists of a single input layer for the focal word and an output prediction layer for contextual words. It formulates the model mathematically as follows. The training goal is to maximize, given the input target word, the conditional log probability of seeing the output contextual word,

$$maxK = max \frac{1}{U} \sum_{u=1}^{U} \log Q(i|x_u) \quad (3)$$

In the neural probabilistic language model, $Q(i|x_u)$ represents the conditional probability and $J$ is the objective function. The gathered and preprocessed test cases are separated into training, validation, and testing subsets in order to have a reliable evaluation of the model and avoid overfitting. In general, training, validation, and testing are commonly split into 70, 15, and 15 percent respectively. The validation set is used to fine-tune hyperparameters and initiate an early halt, whereas the test set is utilized to test the final performance of the Deep-TCP model on unseen data. Such strategy of splitting guarantees recreation and objective evaluation of how the model has prioritized its elements.

### *3.4. Clustering*

Clustering is a valuable unsupervised learning approach that divides unknown things into groups. The members of each category have comparable qualities and characteristics. Clustering relies on word embeddings, which are numerical representations of the text. Clustering is done on purpose before feature extraction to group semantically related test instances, decrease noise and redundancy, and ensure that the next feature extraction procedure learns from homogeneous data distributions. For clustering K-means clustering is utilized.

#### *3.4.1. K-means Clustering*

K-means clustering is used to group similar test case based on the Word embedding, enabling effective organization of testcase. K-means method is robust, quick, and

easy to measure. This is an unsupervised learning strategy for clustering. When data points are properly separated, the K-means method produces excellent results. In this technique, "K" centers are chosen at random or by heuristics and must be separated. After picking data point, assign it to nearest center until no data points remain. Following completion of the first phase, the centers of k clusters are recalculated to generate new "K" centers. Allocate new center and nearest data points. This process will continue until the centers stop migrating. The primary goal of this strategy is to minimize the objective function, as indicated below.

$$fun(Y) = \sum_{v=1}^{d} \sum_{w=1}^{d_v} \|q_v - y_w\|^2 \quad (4)$$

where $\|q_v - y_w\|^2$ denotes the Euclidean distance between $q_v$ and $y_w$."$d_v$" represents number of points in the $u-th$ group, whereas "$d$" represents group centers. The following are the major steps of this algorithm.

Consider $Y = \{y_1, y_2, y_3, \dots, y_d\}$ is a set of group centers, whereas $Q = \{q_1, q_2, q_3, \dots, q_d\}$ represents data items.

Algorithm: K-means Clustering

Step 1: Select d initial cluster centers at random.
Step 2: Measure the distance from each data point to all selected centers.
Step 3: Place every data point into the cluster with the nearest center.
Step 4: Re-estimate the cluster centers by averaging the points assigned to each cluster.

$$y_w = \left(\frac{1}{d_v}\right) \sum_{w=1}^{d_v} q_v$$

Where, "$d_v$" signifies the number of points in the $u-th$ group.
Step 5: Again, compute the distance between each point and the revised cluster centers.
Step 6: Repeat the process until the cluster centers stabilize.

### *3.5. Feature Extraction via Radial basis Function Networks*

Clustering features are retrieved using RBFN to obtain a meaningful feature. RBFN is preferred over autoencoders for feature extraction because it can learn localized, cluster-aware representations with less computational cost. RBFN is one of many types of feedforward neural networks. Neural networks are used to perform mapping functions like $g: S^n \longrightarrow S$ using the following equation:

$$g(z) = X_0 + \sum_{k=0}^{o} X_{k\varphi}(\|Z - d_k\|) \quad (5)$$

where the input vector is represented by $z \epsilon S^n$, the kernel nodes or centers are indicated by $D_k \epsilon S^n$with a range of $1 \leq i \leq m$, the Euclidean distance is represented by $|.|$, the weights are represented by $X_k$ with a range of $1 \leq i \leq o$, and the number of kernel nodes is indicated by $o$.

### *3.6. Convolutional Neural Network-Bidirectional Gated Recurrent Unit*

Convolutional Neural Network-Bidirectional Neural Network is used for testcase prioritize based on high, average and low priority. The CNN-BiGRU architecture was chosen over LSTM or Transformer models because of its fair trade-off between performance, computational efficiency, and scalability. CNN, known for its capacity to capture local characteristics and build spatial hierarchies, has been widely used in test case prioritize for damage detection and lost-data restoration.

CNN's inclusion of local connection, weight sharing, and subsampling methods significantly decreases the number of model parameters, resulting in increased computing efficiency. Convolutional, pooling, and fully linked layers are among the key layers that make up CNN. Stacking multiple layer types can create distinct CNN designs to address specific issues. Convolutional layer is the core of CNN and the major extractor of features to the network. Convolution is a mathematical operation that is used in combining two functions that are real-variable. The following expression is the method:

$$z = \sum_{j=1}^{o} X_j * y + c_j \quad (6)$$

Where $y$ and $z$ signify input and output of convolutional layer, respectively; $X_j$ and $c_j$ represent weights and bias vectors of the $i-th$ filter, respectively; and $'*'$ denotes the convolution operation.

Pooling layer generated through convolution down-sampling of the feature maps, where the local neighbourhoods are aggregated to decrease the spatial dimensions. This reduction aids in managing model complexity, reduces computation and enhances generalization, thus, reducing overfitting. A fully connected layer establishes dense connections between activation of all the neurons and all the activations of the prior layer.

In the normal pipeline, convolutional layers are used to learn hierarchical representations of the features and the pooling layers are used to reduce the dimensionality of the representations. The feature maps obtained are then flattened into a one-dimensional feature vector after multiple convolution-pooling steps and then passed on to fully connected layers to be regressed or classified as needed.

GRU cells construct a GRU layer that takes up sequential data, one time step at a time. The input is processed in sequence using shared weights and biases at every step of a GRU process. The GRU is capable of capturing the important historical context and modeling long-term temporal sequence dependencies by using the current input with a regulated summary of the past states.

GRU cells use two gating processes, namely: update gate and reset gate. When the update gate decides the extent to which the past hidden state should be used in the current state, higher update values preserve more of the past, which is conducive to the continuation of learned temporal characteristics. The reset gate controls the extent to which past information is forgotten during computation of the candidate state; during low reset gate, the prior context is selectively ignored, which enables the GRU to pay attention to novel input patterns where necessary.

GRU's operating method consists of four parts: reset, update, candidate hidden state, and hidden state calculation. The update and reset gates remove long-term dependencies from sequential data, addressing the issue of disappearing gradients. The mathematical formula for these operations is as follows:

$$s_u = \sigma(X_s \cdot [i_{u-1}, y_u] + c_s) \quad (7)$$

$$a_u = \sigma(X_a \cdot [i_{u-1}, y_u] + c_a) \quad (8)$$

$$\widetilde{i_u} = tanh(X_i \cdot [s_u \cdot i_{u-1}, y_u] + c_i) \quad (9)$$

$$i_u = (1-)a_u \cdot i_{u-1} + a_u \cdot \widetilde{i_u} \quad (10)$$

Where $s_u$ and $a_u$ represent the outputs of the reset and update gates, respectively, and $\widetilde{i_u}$ signifies the output of the candidate hidden state. Sigmoid and $tanh$ functions are represented by $\sigma$ and $tanh$, respectively. $i_u$ represents the output information for the current time step. $X_s$, $X_a$, $X_i$, and $c_s$, $c_a$, $c_i$ refer to weight matrices and biases.

The GRU's unidirectional construction, which only transfers neuron states forward, leads to poor usage of sequential input, despite its correlations with both past and future time steps.

This strategy does not fully use the underlying qualities of the data. The BiRNN idea was used to optimize the utilization of past and future data, resulting in the invention of the BiGRU, which is now widely used for response reconstruction and prediction.

BiGRU incorporates a reverse GRU structure into the GRU framework, allowing for recursive feedback of both forward and backward hidden layer states.BiGRU uses dual training to analyze the relationship between present acceleration data and past and future data.

This approach effectively collects data sequence information, improving feature use and missing data reconstruction accuracy. This work uses BiGRU as the fundamental model to effectively extract time-related characteristics from sequential data. The basic equations of BiGRU are shown in Equations (11) to (13):

$$\overrightarrow{i_u} = GRU(y_u, \overrightarrow{i_{u-1}}) \quad (11)$$

$$\overleftarrow{i_u} = GRU(y_u, \overleftarrow{i_{u-1}}) \quad (12)$$

$$i_u = \overrightarrow{X_u}\overrightarrow{i_u} + \overleftarrow{X_u}\overleftarrow{i_u} + c_u \quad (13)$$

The terms GRU and $\overrightarrow{X_u}$ refer to the classic GRU network's training process and forward weight, respectively. Propagation at time$t$, $\overleftarrow{X_u}$ represents the weight associated with backward propagation at time $t$. The bias vector $c_u$ represents the hidden layer's state at time $t$.

Pseudocode: Deep-TCP Model

```
Input:
Test case repository T
    Number of clusters k
    Training epochs E= 100
    Learning rate α = 0.001
Output:
Prioritized test cases (High, Average, Low)
Begin
1.      Extract test cases T from source code repository
2.      Split each test case t_i ∈ T into test steps s_i
3.      For each test step s_i do          //Pre-processing
4.          Apply normalization
5.          Perform tokenization
6.          Apply stemming
7.      End For
8.      Compute word embeddings using Word2Vec for
all test steps     // Word Embedding
9.      Apply K-means clustering on word embeddings
to form k clusters     // Clustering
10.     For each cluster c_j do
// Feature Extraction
11.         Extract features using Radial Basis Function
Network (RBFN)
12.     End For
13.     Initialize CNN-BiGRU model with Adam
optimizer
14.     Train CNN-BiGRU for E epochs with learning
rate α
15.     Classify test cases into priority levels:
        {High Priority, Average Priority, Low Priority}
16.     Return prioritized test cases based on predicted
priority labels
17.     End
```

### *3.7. Design Rationale*

The Deep-TCP framework is designed around three major architectural choices, each of which contributes to effective test case prioritization.

*3.7.1. K-means Clustering for Test Case Organization*

K-means is used to group semantically related test cases based on word embeddings. This grouping helps reduce noise and redundancy, ensures that every cluster contains the same test steps and allows identifying relevant patterns easily. Through aggregating test cases into coherent groups, feature extraction and prioritization can be carried out on more structured data, making it more efficient and predicting accuracy.

*3.7.2. RBFN for Feature Abstraction*

A Radial Basis Function Network (RBFN) is applied after the clustering to obtain discriminatory characteristics. RBFN is well suited to this since its radial basis functions are centered around the cluster prototypes thus learning features locally. In contrast to autoencoders, RBFN produces interpretable and cluster-aware representations with reduced computational complexity, quicker convergence, and reduced overfitting risk, which is best suited to moderate size test data.

*3.7.3. BiGRU for Sequential Test Steps*

Test cases consist of a sequence of stages, and the sequence of actions is decisive. BiGRU captures both forward and backward dependencies efficiently representing temporal interactions, which is used to represent them.

Compared to LSTM or Transformer models, BiGRU is computationally lighter, converges faster, and works well on moderate-sized datasets. When combined with CNN for local feature extraction, the CNN-BiGRU combo efficiently learns both local and sequential patterns, allowing for accurate prioritization.

## 4. Result and Discussion

The experimental results were conducted in Python (PyTorch) and evaluated on a system with NVIDIA RTX 4090 GPU. Deep-TCP model introduced as a PyTorch model and was trained during 100 epochs with a starting learning rate of 0.001 as shown in Table 1. The model performance was evaluated using standard evaluation metrics as accuracy, precision, and recall. Early-stopping technique was used to avoid overfitting and to have stable training. The test case dataset was divided into 70 % training, 15 % validation and 15 % testing with equal distribution of high, medium and low priority. All test steps were preprocessed with normalization, tokenization, and stemming to provide some level of fair comparison; the baseline models (CNN, BiGRU, and CNN-BiGRU) were trained under the same settings. The test cases gathered were well arranged and classified in order to maintain uniformity and avoid confusion in the prioritization.

**Table 1. Training parameters of the proposed deep-TCP model**

| Parameter | Value |
|---|---|
| Epochs | 100 |
| Optimizer | Adam |
| Learning Rate | 0.001 |

### *4.1. Dataset Description*

In this proposed work, the three independent projects-Joda Time, Closure, and Chart-from the respected Defects4j collection [28] are used for preliminary analysis as shown in Table 2. The Defects4j collection has 357 issues and 20,109 tests per project. Every iteration of the project has test cases that correspond to both fixed and problematic code sections. All test cases are written using the JUnit testing methodology.

**Table 2. Information of the dataset**

| Identifier | Project Name | Amount of test Cases |
|---|---|---|
| Chart | Jfreechart | 2,205 |
| Joda Time | Joda-Time | 2,245 |
| Closure | Closure Compiler | 7,927 |

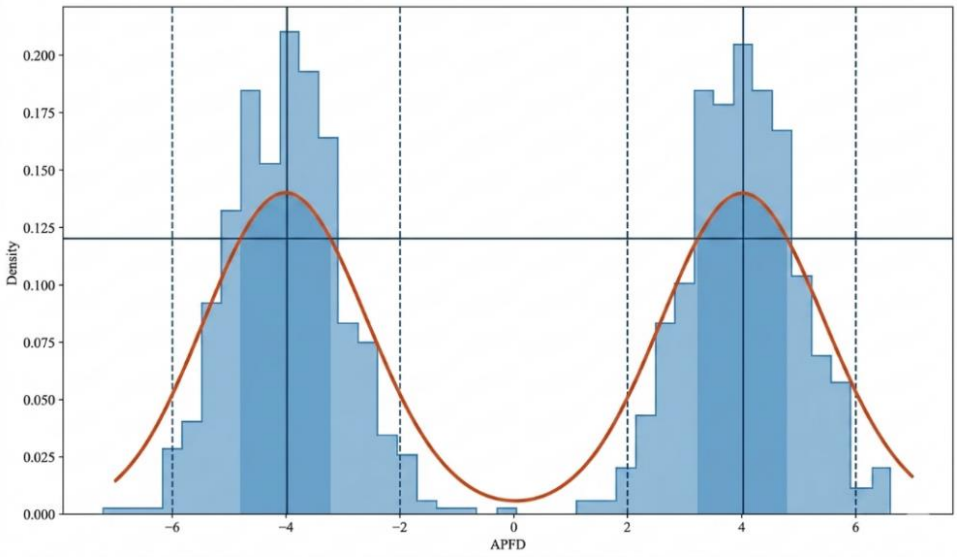


**Fig. 2 Outcomes of the prioritization**

Figure 2 illustrates the outcomes of the prioritization. The x-axis represents as the APFD and y-axis represent as Density. It is evident in Figure 2 that the suggested model generates an average APFD value. Additionally, the histogram peak displays a value of 1, showing a perfect forecast of fresh data. Figure 3 shows APFD comparison of test case. The proposed Deep-TCP techniques are compared with the existing techniques like ATRL-TCP [19], QAOA-TCS [20] and BootQA [22].

Figure 3(a), (b), (c) represent the contrast of APFD for Chart, Joda Time, Closure dataset with proposed and existing techniques. The x-axis represents as Number of Test Cases and y-axis represent as Each Dataset. In closure dataset, APFD value of proposed Deep-TCP techniques for number of testcase 500 is 0.87% and for the existing system are 0.72%, 0.70%, 0.78% respectively. Likewise, the APFD value for the proposed techniques is higher than existing techniques for all dataset.

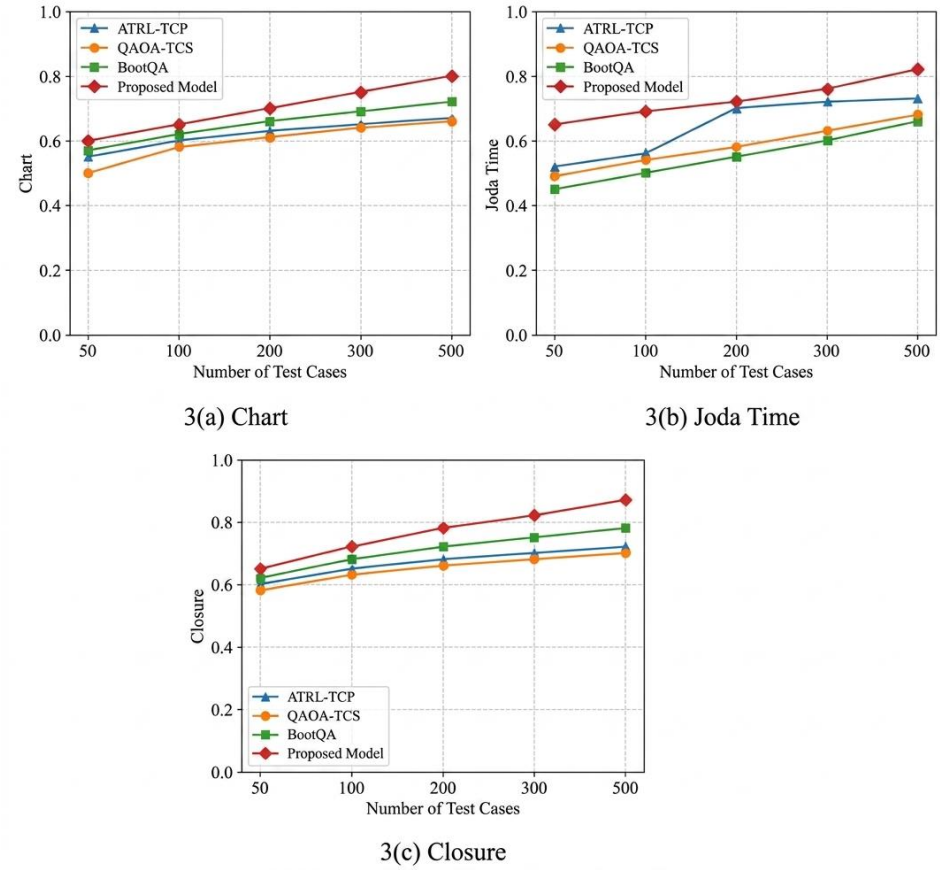


**Fig. 3 APFD comparison of test cases**

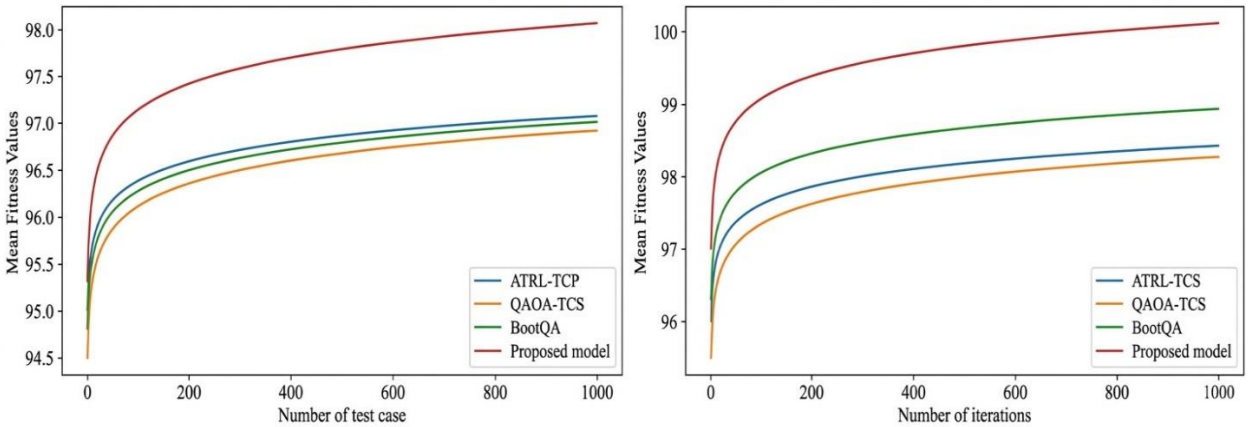


**Fig. 4 Convergence curves of testcase**

Figure 4 represent the Convergence curves of Testcase. Figure 4(a), (b) represents the contract of mean fitness value for Fault coverage and statement coverage with proposed and existing techniques. The x-axis represents as Number of Test Cases and y-axis represent as Mean Fitness Values. The proposed Deep-TCPtechniques has higher coverage for both the fault and statement coverage than the existing techniques. In fault coverage, mean fitness value of the proposed Deep-TCS techniques for number of Testcase 600 is 97.58% and the existing system are 96.57%, 96.45%, 96.50% respectively. In statement coverage, mean fitness value of the proposed Deep-TCP techniques for number of Testcase 600 is 99.95% and the existing system are 97.89%, 97.50%, 98.45% respectively.

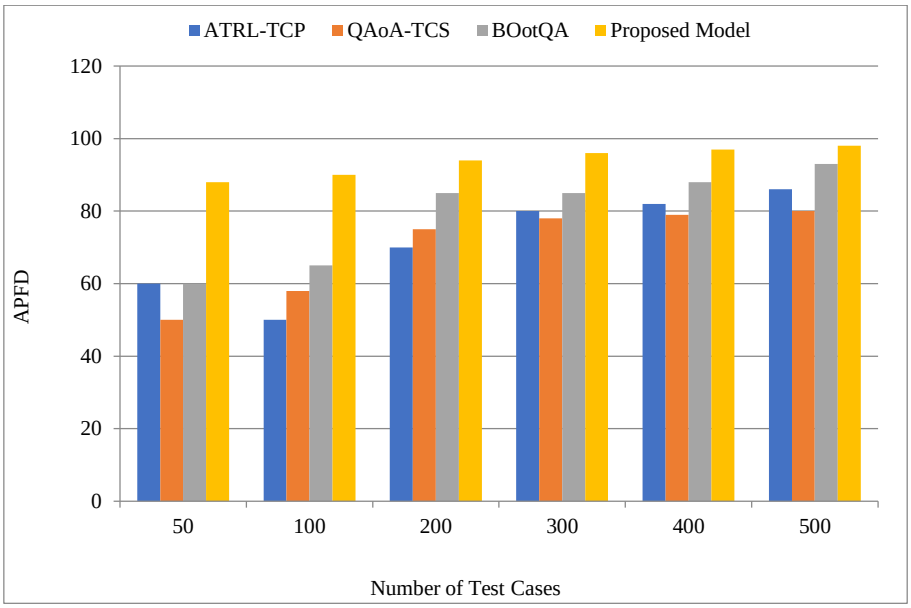


**Fig. 5 The average APFD of existing and proposed technique for prioritizing**

Figure 5 illustrates the average APFD of existing ATRL-TCP [19], QAOA-TCS [20] and BootQA [22] and ProposedDeep-TCP techniques for Testcase prioritizing. The x-axis represents the number of test cases, whereas the y-axis represents APFD. The proposed Deep-TCP techniques are higher than the existing techniques.

For instance, the proposed Deep-TCP model produced a APFD of 99% with 500 testcases, compared to 86%, 80% and 93% for the ATRL-TCP, QAOA-TCS and BootQA approaches. The proposed model is 94.5%, which higher than the ATRL-TCP of 71.25%, QAOA-TCS of 73.5%, BootQA of 81.5% respectively.

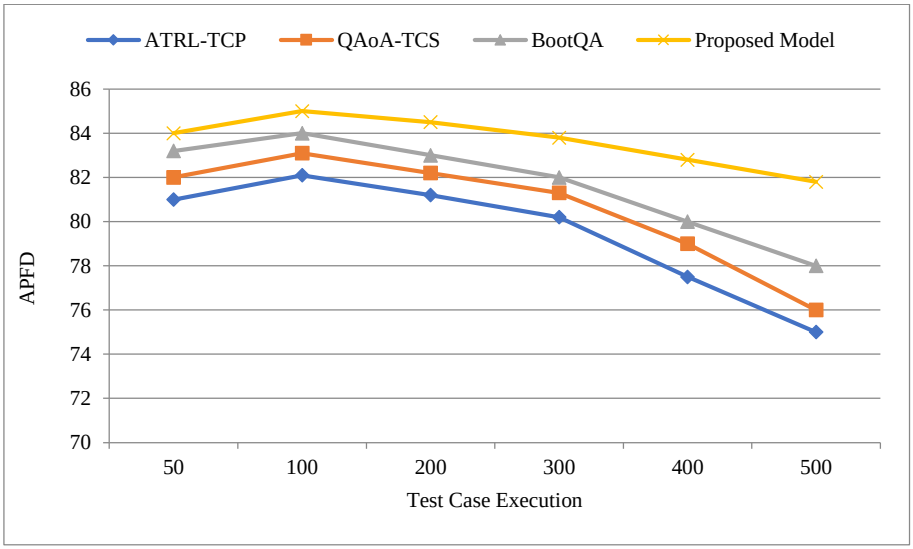


**Fig. 6 APFD trend of proposed and existing method**

Figure 6 display the APFD trend of proposed Deep-TCP method and the existing techniques such as ATRL-TCP [19], QAOA-TCS [20] and BootQA [22]. X-axis represent the Testcase Execution and y-axis represent the APFD values. As compared to the existing methodology, the proposed Deep-TCP method has higher APFD up to 84%, and the lower APFD up to 75%. When the number of Testcase execution is 300, the corresponding APFD values computed by the existing ATRL-TCP, QAOA-TCS, BootQA and Proposed Deep-TCP are 80.2%, 81.3%, 82.0%, and 83.8% respectively. When the number of Testcase execution is 500, the corresponding APFD values computed by the existing ATRL-TCP, QAOA-TCS, BootQA and Proposed Deep-TCP are 75%, 76%, 78.0%, and 81.8% respectively.

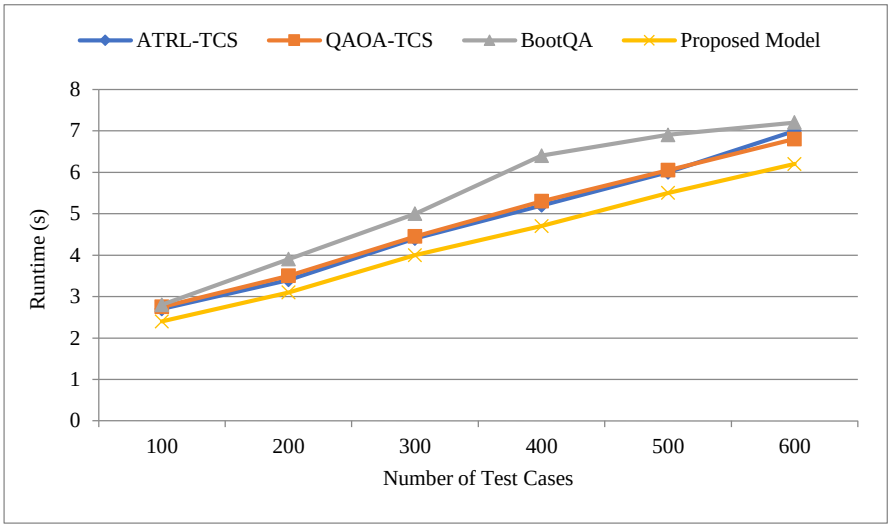


**Fig. 7 Runtime comparison for existing and proposed techniques**

Figure 7 shows the runtime comparison of existing and proposed approaches, including ATRL-TCP [19], QAOA-TCS [20] and BootQA [22], and the proposed Deep-TCP model. The x-axis represents as Number of Test Cases and y-axis represent as Runtime. Further demonstrates how runtime increase as the number of Testcase increase. When a Testcase is 300, the runtime of the proposed methods is 4.7 s and the existing ATRL-TCP, QAOA-TCS and BootQA yield 5.15 s, 5.3, and 6.4 s respectively. As the number of Testcase increase, so does the amount of runtime comparison.

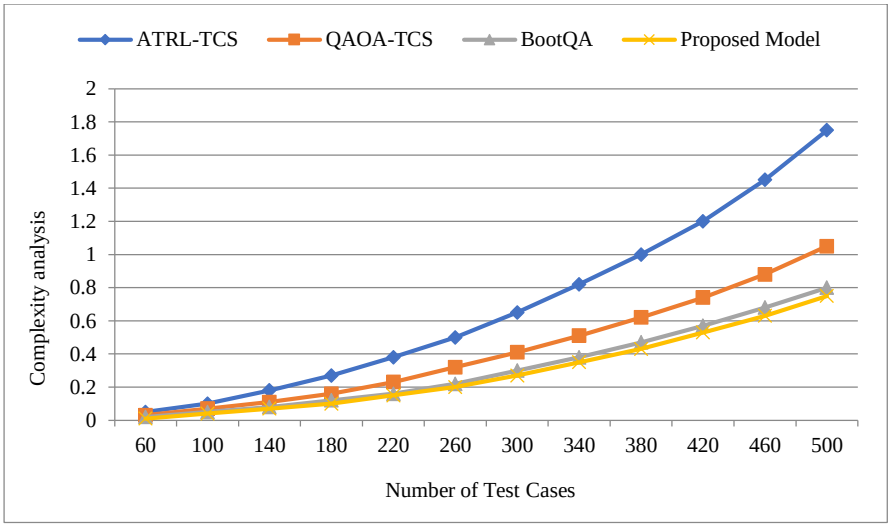


**Fig. 8 Complexity analysis for existing and proposed techniques**

Figure 8 illustrates the complexity analysis for existing and proposed techniques such as ATRL-TCP [19], QAOA-TCS [20] and BootQA [22] and proposed Deep-TCP model. The x-axis represents the Number of Testcase and y-axis represent the Complexity analysis. For instance, the proposed Deep-TCP model produced a complexity of 0.27 s with 300 Testcases, compared to 0.65 s, 0.41 s and 0.30 s for ATRL-TCP, QAOA-TCS and BootQA approaches. The proposed

Deep-TCP strategy eventually achieves a better complexity with 500 Testcases, while the existing ATRL-TCP, QAOA-TCS and BootQA strategies provided a higher complexity.

**Table 3. Comparison of statistical testing for proposed and existing approaches**

| Author | Methods | Accuracy | p-value |
|---|---|---|---|
| Su, Q., | ATRL-TCP | 92.80% | 0.041 |
| Trovato, A., | QAOA-TCS | 91.10% | 0.019 |
| Wang, X., | BootQA | 89.70% | 0.028 |
| Proposed | Deep-TCP | 99.03% | 0.007 |

Table 3 indicates the comparison of statistical significance for proposed and other existing approaches like ATRL-TCP [19], QAOA-TCS [20], BootQA [22]. The Deep-TCP model achieves an accuracy of 99.03% compared to other models obtained like ATRL-TCP (92.80%) [19], QAOA-TCS (91.10%) [20], and BootQA (89.70%) [22] respectively. Additionally, the Deep-TCP model attains a low p-value of 0.007 shows strong statistical significance and improved reliability compared to the existing methods.

Figure 9 depicts the suggested model's performance in comparison to the current ATRL-TCP [19], QAOA-TCS [20] and BootQA [22] and proposed Deep-TCP model. The x-axis represents as performance Metrics and y-axis represent as Values. The proposed model achieved remarkable results, with precision of 98.85%, recall of 98.64%, F-measure of 98.95%, and accuracy of 99.03%, whereas the other remaining classifiers achieved approximate rates of precision, recall, and F-measure of 93%, 95%, and 94%respectively. Figure 6 shows that the proposed model outperforms earlier models due to its capacity to optimize the learning process, which leads to better overall performance.

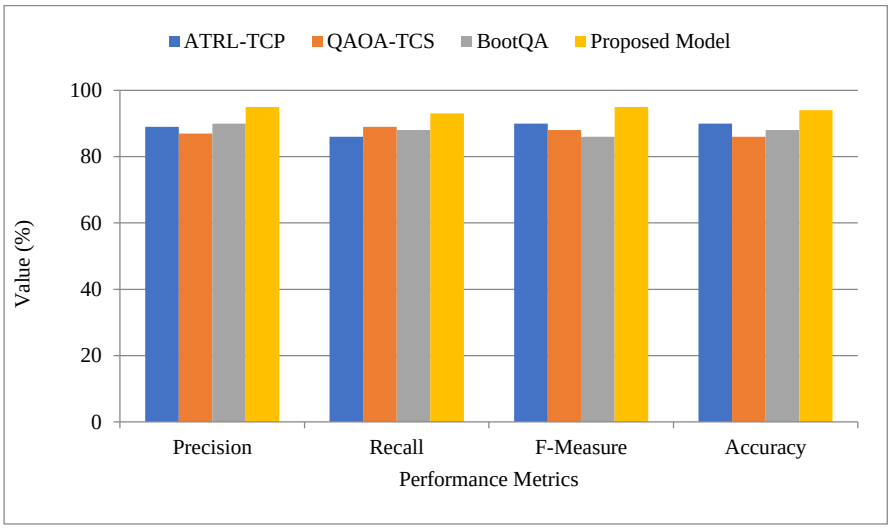


**Fig. 9 Performances metrics for proposed and existing techniques**

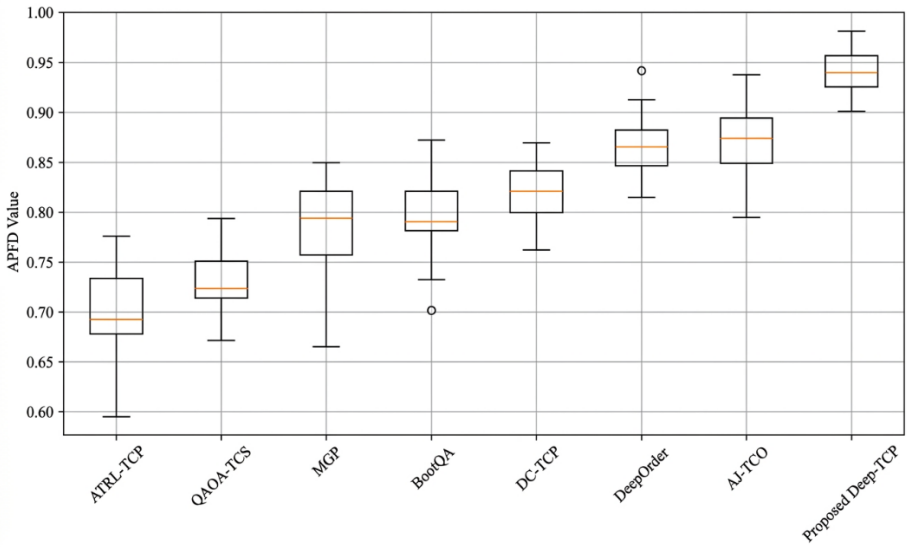


**Fig. 10 Comparison of APFD performance for TCP methods**

Figure 10 indicates that suggested Deep-TCP model will be assessed based on the APFD measure and will be compared to the current TCP approaches like ATRL-TCP, QAOA-TCS, HGP, BootQA, DC-TCP, DeepOrder and AI-TCO. The findings indicate that the model suggested has a better APFD value, which means that it is efficient in terms of fault diagnosis at an earlier stage and test case priority than current methods.

**Table 4. Ablation result for deep-TCP accuracy**

| Techniques | Accuracy (%) |
|---|---|
| With Clustering | 98.15 |
| Without Clustering | 96.50 |
| With RBFN | 98.25 |
| Without RBFN | 96.00 |
| CNN (only) | 97.10 |
| BiGRU (only) | 97.50 |
| CNN + BiGRU | 98.80 |
| Proposed Deep-TCP | 99.03 |

Table 4 demonstrates the methods that use clustering and RBFN exhibit better accuracy than the similar methods that do not use these elements. The CNN-BiGRU combination is an additional improvement, which allows to capture the locals and sequential elements of the test steps. On the whole, the Proposed Deep-TCP model has the best accuracy of 99.03 exceeding all the single and composite baseline methods which shows its ability to prioritize automated test cases with high efficiency and accuracy.

**Table 5. Statistical analysis of prioritization techniques**

| Compared Models | Accuracy (%) | p-value |
|---|---|---|
| CNN | 97.10 ± 0.18 | 0.015 |
| BiGRU | 97.50 ± 0.16 | 0.012 |
| CNN + BiGRU | 98.80 ± 0.14 | 0.004 |
| Proposed Deep-TCP | 99.03± 0.10 | 0.001 |

As shown in Table 5, the proposed Deep-TCP model achieves a mean accuracy of 99.03% with a low standard deviation of ±0.10, indicating highly stable performance across multiple runs. The p-value of 0.001 which goes with the latter indicates that the over-base line techniques are significantly better.

Although CNN and BiGRU are the only two methods which also demonstrate significant improvements with p-values of 0.015 and 0.012 respectively, their accuracy is not the same as that of CNN-BiGRU and Deep-TCP methods. The findings indicate that the proposed Deep-TCP framework has strong and effective test case prioritization, which is more efficient than all techniques compared.

Table 6 show that the proposed Deep-TCP model has better APFD and classification performance than the current baseline methods, and shows greater performance in terms of early fault detection and test case prioritization.

**Table 6. Performance comparison with recent baseline methods**

| Method | APFD | Accuracy (%) | Precision (%) | Recall (%) |
|---|---|---|---|---|
| ATRL-TCP | 0.72 | 85.1 | 84.6 | 83.9 |
| DeepOrder | 0.86 | 92.1 | 91.5 | 90.6 |
| AI-TCO | 0.88 | 93.2 | 92.6 | 91.7 |
| Proposed Deep-TCP | 0.94 | 99.03 | 95.7 | 94.8 |

## 5. Conclusion

In this section, a novel An Intelligent Deep Learning Model for Automated Test Case Prioritization Systems using Neural Networks (Deep-TCP) has been proposed. Testcases have been extracted from the source code repository and broken down into test stages. To eliminate noise, testing steps include pre-processing with techniques like as normalization, tokenization, and stemming. Following pre-processing, word embedding is calculated and test cases are clustered using K-means clustering, and. Radial Basis Function Network (RBFN) is utilized to extract significant characteristics, while Convolutional Neural Network-Bidirectional Gated Recurrent Unit (CNN-BiGRU) is used to prioritize test cases as high, average, or low. The average APFD of the proposed method is 94.5% which is higher than 71.25% at ATRL-TCP, 73.5%% at QAOA-TCS and 81.5% at BootQA approaches. The proposed Deep-TCP model is practical to implement in the continuous integration setting in order to automatically rank test cases in the regression process. The framework is also scalable and can support large test case repositories of more complicated software projects.

The model makes testing more efficient by allocating more critical test cases earlier which allows earlier fault detection. Nevertheless, the efficiency of the suggested methodology relies on the quality and diversity of the gathered test case data that can influence the quality of prioritization in various software systems.

Future research will improve this framework by using transformer-based embeddings, adaptive clustering approaches, and continuous learning processes to improve accuracy, scalability, and real-world applicability in automated test case prioritizing.

## Conflicts of Interest

The authors declare that there is no conflict of interest.

## Funding Statement

To support this research, the authors did not get any funds.

## Acknowledgments

The author would like to express his heartfelt gratitude to the supervisor for his guidance and unwavering support during this research for his guidance and support.

## References


[1] Syed Roohullah Jan et al., "An Innovative Approach to Investigate Various Software Testing Techniques and Strategies," *International Journal of Scientific Research in Science, Engineering and Technology*, vol. 2, no. 2, pp. 682-689, 2016. [Google Scholar]

[2] Abhijit A. Sawant, Pranit H. Bari, and P.M. Chawan, "Software Testing Techniques and Strategies," *International Journal of Engineering Research and Applications*, vol. 2, no. 3, pp. 980-986, 2012. [Google Scholar]

[3] Mohd. Ehmer Khan, "Different forms of Software Testing Techniques for Finding Errors," *International Journal of Computer Science Issues*, vol. 7, no. 3, pp. 11-16, 2010. [Google Scholar]

[4] Albin Lönnfält et al., "An Intelligent Test Management System for Optimizing Decision Making During Software Testing," *Journal of Systems and Software*, vol. 219, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[5] Muhammad Faisal Abrar et al., "A Data-Driven Analysis of Software Testing Automation Challenges using Structural Equation Modeling (SEM) Approach," *IEEE Access*, vol. 13, pp. 96159-96181, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[6] Mark Harman, Peter O'Hearn, and Shubho Sengupta, "Harden and Catch for Just-in-Time Assured LLM-based Software Testing: Open Research Challenges," *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, Association for Computing Machinery, New York, NY, United States, pp. 1-17, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[7] Zheng Li, Mark Harman, and Robert M. Hierons, "Search Algorithms for Regression Test Case Prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225-237, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[8] 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*, vol. 25, no. 7, pp. 5163-5172, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[9] Ahmadreza Saboor Yaraghi et al., "Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1615-1639, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[10] Mohammed Assiri, "Test Case Prioritization using Dragon Boat Optimization for Software Quality Testing," *Electronics*, vol. 14, no. 8, pp. 1-20, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[11] Tapas Kumar Choudhury et al., "AnoLSTM: A Deep Learning Approach for Test Cases Prioritization," *Procedia Computer Science*, vol. 258, pp. 1793-1803, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[12] Aishwaryarani Behera, and Arup Abhinna Acharya, "An Effective GRU-based Deep Learning Method for Test Case Prioritization in Continuous Integration Testing," *Procedia Computer Science*, vol. 258, pp. 4070-4083, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[13] Jeongki Son et al., "Evaluating Machine Learning-based Test Case Prioritization in the Real World: An Experiment with SAP HANA," *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Napoli, Italy, pp. 522-532, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[14] Andreea Vescan, and Cristina-Maria Tiutin, "Test Case Prioritization based on Neural Networks Classification: A Replication Study and Hyper-Parameter Optimization using Taguchi Methods," *IEEE Access*, vol. 13, pp. 118082-118095, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[15] Abdallh Mostafa Mohamed Mohamed Menshawy, and Mohammad Nasar, "Optimizing Software Quality: Integrating Test Case Prioritization, Defect Prediction, and Resource Allocation Strategies," *East Journal of Computer Science*, vol. 1, no. 2, pp. 16-21, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[16] Hemant Kumar, and Vipin Saxena, "Optimization and Prioritization of Test Cases through the Hungarian Algorithm," *Journal of Advances in Mathematics and Computer Science*, vol. 40, no. 3, pp. 61-72, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[17] Abdulkarim Bello, and Hauwa Muhammed Alhassan, "Cost-Cognizant Test Case Prioritization for Software Product Line using Genetic Algorithm," *FUDMA Journal of Sciences*, vol. 9, no. 9, pp. 129-138, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[18] Shounak Rushikesh Sugave et al., "Fault-Aware Test Case Prioritization in Software Testing using Jaya Archimedes Optimization Algorithm," *Journal of Electronic Testing*, vol. 41, no. 1, pp. 41-61, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[19] Qingran Su et al., "Attention Transfer Reinforcement Learning for Test Case Prioritization in Continuous Integration," *Applied Sciences*, vol. 15, no. 4, pp. 1-22, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[20] Antonio Trovato, Martin Beseda, and Dario Di Nucci, "A Preliminary Investigation on the Usage of Quantum Approximate Optimization Algorithms for Test Case Selection," *Proceedings of the 2025 29th International Conference on Evaluation and Assessment in Software Engineering Companion*, Association for Computing Machinery, New York, NY, United States, pp. 56-60, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[21] Christoph Laaber, Tao Yue, and Shaukat Ali, "Evaluating Search-based Software Microbenchmark Prioritization," *IEEE Transactions on Software Engineering*, vol. 50, no. 7, pp. 1687-1703, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[22] Xinyi Wang et al., "Test Case Minimization with Quantum Annealers," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 1, pp. 1-24, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Zikang Zhang et al., "Exploiting DBSCAN and Combination Strategy to Prioritize the Test Suite in Regression Testing," *IET Software*, vol. 2024, no. 1, pp. 1-14, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[24] Aizaz Sharif, Dusica Marijan, and Marius Liaaen, "DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing," *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Luxembourg, pp. 525-534, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[25] Ada John, Isreal John, and Trump Dion, "Integrating AI-Driven Test Case Optimization into Continuous Integration/Continuous Delivery (CI/CD) Pipeline Authors," *Continuous Delivery (CI/CD) Pipeline Authors*, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[26] Zheng Li et al., "An Environment Adaptation Agent of Reinforcement Learning in Continuous Integration Test Case Prioritization," *International Journal of Software Engineering and Knowledge Engineering*, vol. 36, no. 2, pp. 311-341, 2026. [CrossRef] [Google Scholar] [Publisher Link]

[27] Ajmer Singh et al., "A Systematic Literature Review on Test Case Prioritization Techniques," *Agile Software Development*, pp. 101-159, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[28] Allan Mori, Ana C.R. Paiva, and Simone R.S. Souza, "Code Change and Smell Techniques for Regression Test Selection," *Software Quality Journal*, vol. 33, no. 1, pp. 1-24, 2025. [CrossRef] [Google Scholar] [Publisher Link]