Explainable Test Case Prioritization in Continuous Integration through Incremental Learning Approach

Kamal Garg
PhD Scholar
GLA University, Mathura
kamal.mtr@gmail.com

Rohit Agarwal
Professor, CEA
GLA University, Mathura
rohit.agrwal@gla.ac.in

Shashi Shekhar
Professor, CSE
Amity University, Patna
shashi.shekhar@gmail.com

2024 International Conference on Communication, Control, and Intelligent Systems (CCIS) | 979-8-3315-2820-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCIS63231.2024.10932075

***Abstract*— In Continuous Integration (CI) environments, where software undergoes frequent updates, regression testing is vital in ensuring software quality. However, rerunning every test case becomes impractical with the constant changes. Test Case Prioritization (TCP) addresses this issue, and Machine Learning (ML) is increasingly used to manage regression testing. However, many ML models struggle to adapt to new features or changes in CI and lack transparency, complicating the regression process. To address these issues, we propose an incremental learning-based explainable ML model for TCP in CI environments, which adaptively incorporates new changes. We use SHapley Additive exPlanations (SHAP) to evaluate feature contributions and help testers understand the model's functionality. Our model is trained and tested on 20 open-source software projects. Its performance is assessed using Accuracy and F1 Score, while test case prioritization is evaluated with the Average Percentage of Faults Detected (APFD) and a new metric, the Failed Test Ranking Score (FTRS).**

Keywords*— *Continuous Integration, Deep Learning Model, Regression Testing, Software Testing, Test case Prioritization.

I. INTRODUCTION

Software testing is essential and critical in software development, especially within a continuous integration (CI) environment. CI involves regularly integrating and testing changes as they are made rather than waiting until the end of the development cycle [1]. Many approaches are used during software testing in CI environments, but TCP is particularly popular. In CI, developers must execute many test cases quickly, and prioritization helps focus efforts on critical cases, reducing the time and effort required for regression testing. This improves time and resource management, leading to quicker defect resolution and a more efficient testing process, ultimately enhancing software quality in a CI environment.

From greedy algorithms to metaheuristic approaches, various methods are used for test case prioritization [2,3]. However, ML has become increasingly popular due to its ability to handle large amounts of historical data and its capacity to identify patterns within datasets. Figure 1 shows the basic block diagram of how test case prioritization can be performed using the ML model in the CI environment. The process begins with developers writing code and committing changes to a code repository. The CI server monitors this repository and triggers the build process upon detecting changes. During the build process, the code is compiled to ensure that new changes integrate seamlessly with the existing codebase. Initial unit tests are then generated to verify that individual software components function correctly. Following this, a feature extraction occurs, where critical aspects of the code and test cases are identified and stored in a data repository for further analysis. The data repository, which stores the extracted features and the results of test case executions, is used to train the ML model for TCP over time. The code is deployed once the testing phase concludes successfully, with all prioritized test cases passing and any identified bugs resolved.

A key contribution of the proposed approach is that it enables incremental learning to improve the testing process continuously, exemplifying how regression testing is integrated into a CI environment. Critical parts of the software are tested first by prioritizing test cases, leading to faster identification and resolution of defects and resulting in more stable and reliable software deployments. This method accelerates the testing cycle and ensures that the most crucial functionalities are verified, thereby maintaining high software quality standards. Another key contribution is the addition of interpretability and explainability to the ML model used for TCP, providing clearer insights into the prioritization decisions.

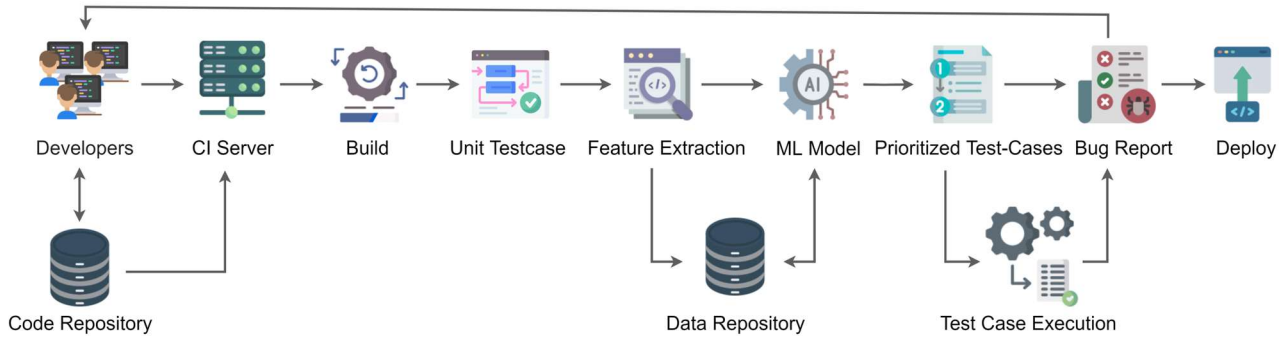

Fig. 1. Deployment of continuous integration and testcase case prioritization in software development

II. Related Study

This related study focuses on Continuous Integration (CI) and Regression Testing in software development and the challenges associated with managing and maintaining test cases within CI environments [4]. TCP helps improve the testing process's efficiency and effectiveness by identifying and executing the most critical test cases [5]. Researchers use ML models to provide various test selection, minimization, and prioritization approaches [6]. These models can adapt to different testing environments and scale effectively to handle large and complex test suites in modern software development practices. These models include supervised, unsupervised, reinforced, hybrid, and deep learning approaches [7] [8]. Marijan D. [9] presents a detailed comparative study of ML-based TCP in the context of CI. This work evaluates the accuracy of models for fault detection rate. It compares the performance of four models: support-vector machines, artificial neural networks, gradient-boosting decision trees, and Lambda Rank using NN (LRN), where LRN gives the most promising results.

TCP includes various innovative approaches, such as rule-based, clustering-based [7], dispersity-based, cost-based, coverage-based [10], learning-based, history-based, and hybrid approaches. Managing large test suites necessitates scalable approaches to mitigate the costs of repeated testing. For instance, the authors in [11] propose a CI and regression testing framework in IoT-based systems using supervised deep learning algorithms that achieve 92% accuracy. In the other study [12], Recurrent Neural Network (RNN)-based models like the Gated Recurrent Unit (GRU) are employed to analyze the QoS for regression testing. Quality of Service (QoS) metrics, including reliability and scalability, are pivotal in assessing web service performance.

It is observed that the neighboring test cases have a degree of similarity while dispersed test cases have dissimilarity. Dispersity-based prioritization [13] uses the concept of dispersed test cases to calculate the distance. It is a technique used in software testing to prioritize test cases based on their level of diversity. In dispersity-based prioritization, the distance between two test cases is calculated based on their level of dissimilarity. Test cases that are more dissimilar are considered further apart, while more similar test cases are considered closer together. In traditional random testing, test cases are generated randomly and executed without any particular order or priority. However, this results in redundant or unnecessary test cases, which are time-consuming and costly. The author in [14] proposes Mirror Adaptive Random Testing (MART), which aims to improve the efficiency of random testing by reducing redundant test cases.

The existing TCP approaches for CI optimize for only limited historical test cases. One advantage of end-to-end DNNs [15] for TCP is that they potentially capture complex relationships between the test cases and their features, which traditional TCP techniques may not capture. Another advantage is that the model trains using large datasets, which improves the model's accuracy. However, DNN models are computationally expensive and require significant computing resources. Deep-order [16] addresses scalability by prioritizing historical test cases using Convolutional Neural Networks (CNNs) to evaluate test case importance based on source code and metadata such as their execution time and coverage. DeepOrder is a deep learning-based approach for TCP, which ranks and orders test cases based on their importance. It uses a CNN to learn a feature representation of test cases based on their source code and associated metadata, such as their execution time and coverage. In the next section, the proposed methodology and procedure are discussed.

III. Proposed Methodology

Identifying and designing a machine learning model for Test Case Prioritization in Continuous Integration environments is challenging due to the rapid and frequent code changes. Incremental learning (IL) has been identified as the best approach to address this problem, as it is well-suited for environments where data arrives incrementally, such as new builds in a CI pipeline. With IL, the model is updated with each new batch of data rather than being retrained from scratch, allowing it to adapt to new changes without losing previously acquired knowledge, a concept known as "Learning without Forgetting" [17]. Extensive research has highlighted the importance of enabling ML models to learn new information while retaining prior knowledge. Studies have introduced strategies like online continuous training for support vector machines and ensemble-based approaches for neural networks to address incremental learning challenges. These studies collectively underscore the efficacy of incremental learning in enhancing the adaptability and performance of ML models in dynamic environments.

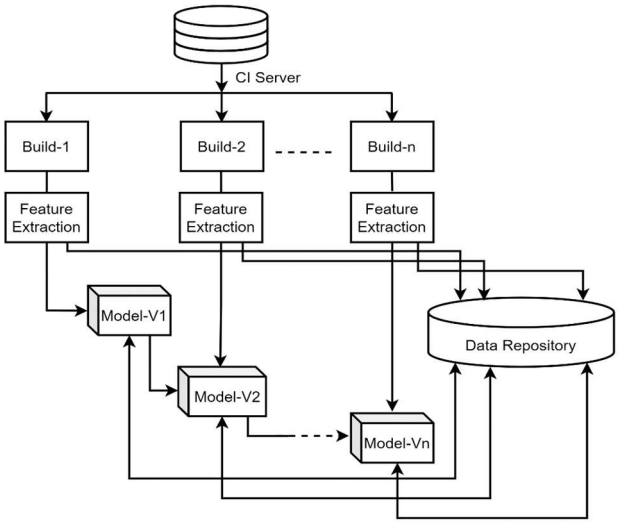

Fig. 2. Integration of incremental learning in CI environment

Figure 2 illustrates the workflow of incremental learning in continuous integration builds. At each build cycle, the CI server generates a software build called Build-k. Features are extracted from Build-k and stored in a data repository. Model-Vk (ML model version k) is then trained on these extracted features, and the model's settings or hyperparameters are also stored in the data repository. For the subsequent build, Build k+1, new features are extracted and stored in the data repository. Instead of training the model from scratch, we utilize the previously trained model, Model-Vk, with its corresponding settings or parameters stored in the data repository as a starting point. This model is further trained on the new dataset to create an updated model, Model-Vk+1. This process is repeated for each new build, generating an updated model on a new dataset without losing information from the previous updates. This approach optimizes software testing through dynamic adaptations to new data, accelerating development cycles and speeding up the delivery of stable software builds.

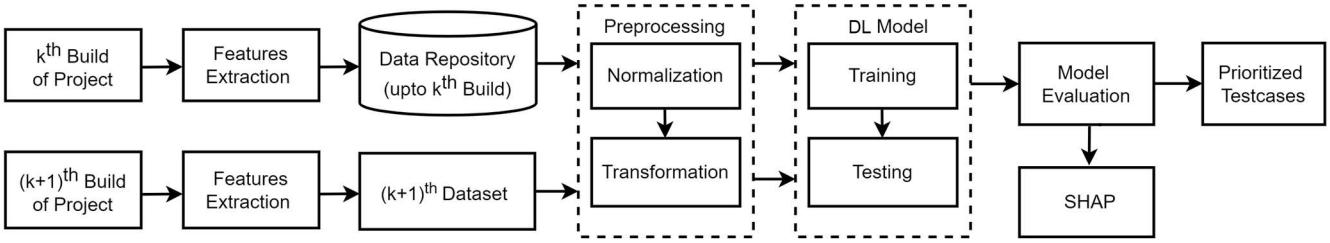

Fig. 3. Block diagram of proposed approach

Figure 3 depicts the proposed Deep Learning (DL) based TCP model where the CI project evolves with each cycle k, introducing new modules or modifying existing ones, generating fresh raw data, including code coverage metrics, test case histories, and building execution logs. Data extraction from these sources often necessitates using various tools or methods. Since raw data like build logs contain detailed information that is not directly relevant to test case properties, it is crucial to aggregate and reformat this data to derive desired features. These restructured features from each build are stored in a data repository for subsequent model training and testing.

Before training the ML model, the training dataset undergoes preprocessing, which involves transforming categorical input data, labeling output data, and normalizing the dataset. During transformation, the dataset includes data from CI builds (build-1 to build-k), each characterized by 150 features and two output values. The model uses a dual-output format to separately analyze fault predictions and test case priorities, enhancing analytical depth. The first output label (test case verdict) is extracted directly from the log files. In contrast, the second output (initial test case priority) is generated using a weighted ranking method based on criteria defined by the software tester. This prioritization process allows for a nuanced and customizable approach to TCP, ensuring that critical test cases are addressed first. To understand how the DL model performs based on user-defined criteria for test case prioritization, we use explainable AI (XAI) methods such as Shapley Additive exPlanations (SHAP) [18]. These techniques provide insights into the decision-making process and visualize relevant input features, enhancing the interpretability and transparency of the model's decisions.

The proposed solution integrates a 1D Convolutional Neural Network (1DCNN) to tackle the TCP problems. A 1DCNN is chosen for its suitability and direct applicability to tabular datasets. The model is trained using data compiled from CI builds up to the kth instance, and its generalizability is tested using data from the subsequent (k+1)th build.

IV. Experimental Evaluation

Twenty (20) open-source projects were used to create an efficient ML model for TCP. First, a CI environment was set up. Relevant features were then extracted from the project repository through statistical analysis of source code, version control history, build logs, and test case execution records. These features were grouped into nine categories based on their properties. More details about the dataset can be found in a GitHub repository [21], which explains the procedures for extracting features and data samples from raw project data.

Once the initial model is trained on the first 9 builds, we evaluate its performance on the 10th build in the sequence. Consequently, we use builds 11 to 19 as the new training dataset for updating the previously trained model and then test the updated model on the 20th build. This iterative process follows an incremental learning approach, with a build window size of 10, allowing us to train the model again on new datasets and fine-tune hyperparameters without losing valuable information from the previous training.

The experiments were conducted on a machine with an Intel i5 processor and 16 GB memory in a Python 3.10.8 environment.

A. Evaluation Metrics

To evaluate the proposed ML-based TCP models, which predict both the verdict and priority of test cases, Mean Square Error (MSE) is used for priority, and Accuracy and F1 Score are used for verdicts. The F1 Score provides a more robust evaluation alongside accuracy. The objective of a TCP algorithm is to order test cases to identify failures early, measured by the APFD [19] and APFDc [20] metrics. APFD measures the effectiveness of test prioritization, while APFDc accounts for both failures and execution time. For further or more accurate TCP analysis, we introduce a new metric, FTRS.

B. Failed Test Ranking Score (FTRS):

The primary objective of a TCP algorithm revolves around prioritizing test cases to ensure that those likely to fail are executed earlier during the testing process. A novel metric called the FTRS is proposed to measure the success of this objective. In this approach, test case prioritization is based on aligning with the optimal order of test cases, ensuring that all failed test cases are executed before any passed ones, and considering each failure to represent a distinct fault in the SUT. Equation 1 describes the FTRS value.

$$\mathrm{FTRS} = \frac{\sum_{i=1}^{n} n}{\sum_{i=1}^{n} M} \times \frac{k}{n} \qquad (1)$$

Here, n is the total number of failed test cases. So $\sum_{i=1}^{n} n$ refer to the sum of the top n position of failed test cases in the optimal test suite. M describes the position of failed test cases in the prioritized test suite. So $\sum_{i=1}^{n} M$ refers to the sum of the position of failed test cases in the prioritized test suite, and k represents the number of failed test cases within the top n position of the optimal test suite.

V. Result And Analysis

This section is divided into three parts. In the first part, the performance of the proposed 1DCNN models is compared with XGboost (XGB) and Logistic Regression (LR) using various metrics described in the previous section. In the second part, feature analysis is performed. In the last section, XAI methods explain how the proposed model works.

TABLE I. COMPARATIVE ANALYSIS OF ML MODELS PERFORMANCE

	Projects	XGB	LR	1DCNN
Accuracy	Range	0.884 - 0.998	0.807 - 0.991	0.887 - 0.998
	Average	0.968135	0.93185	0.97385
F1 Score	Range	0.861 - 0.968	0.732 - 0.953	0.861 - 0.984
	Average	0.93215	0.88135	0.95225
RMSE	Range	0.039 - 0.108	0.002 - 0.169	0.040 - 0.172
	Average	0.10879	0.122285	0.11124

The comparative analysis of TCP models (Table I) reveals that the 1DCNN excels in both accuracy and F1 Score, with averages of 0.97385 and 0.95225, respectively, indicating its robustness in correctly identifying test case verdicts with balanced precision and recall. On the other hand, LR shows the worst performance, while XGB presents a balanced performance with an average accuracy of 0.968135, an F1 Score of 0.93215, and the lowest average RMSE value of 0.10879. Therefore, 1DCNN is ideal for scenarios demanding high accuracy and F1 Score, whereas XGB offers a well-rounded option as a general approach.

TABLE II. COMPARATIVE ANALYSIS OF ML MODELS IN TERMS OF TCP.

	Projects	XGB	LR	1DCNN
APFD	Range	0.67 - 0.999	0.532 - 0.999	0.853 - 0.999
	Average	0.920985	0.895005	0.9656
APFDc	Range	0.627 - 0.93	0.641 - 0.94	0.641 - 0.989
	Average	0.71415	0.7033	0.72105
FTRS	Range	0 - 1	0 - 1	0.41 - 1
	Average	0.6734	0.50805	0.9087

Table II focuses on the performance of ML models in terms of TCP order. For APFD, the 1DCNN model outperforms the others with an average of 0.9656, indicating its superior capability in prioritized failed test cases. XGB and LR follow with averages of 0.920985 and 0.895005, respectively, showing that while they are effective, they do not match the efficiency of 1DCNN. Regarding the APFDc, 1DCNN again leads slightly with an average of 0.72105, followed by XGB at 0.71415 and LR at 0.7033, demonstrating that 1DCNN maintains its effectiveness even when costs are considered. Finally, for the FTRS, 1DCNN exhibits a significantly higher average of 0.9087, showcasing its resilience to test case failure, whereas XGB and LR have averages of 0.6734 and 0.50805, respectively, indicating less effectiveness in TCP. Regarding FTRS, once again, 1DCNN performs well compared to others. The minimum FTRS value of 0.41 indicates that 1DCNN always includes some test cases in the top order of prioritized test cases. In contrast, the FTRS minimum value of 0 for XGB and LR indicates that, in some cases, they are unable to maintain failed test cases in the top orders of test cases.

TABLE III. TOP 10 IMPORTANT FEATURES IDENTIFIED BY TCP MODELS

Sn	XGB	1DCNN	LR
1	REC_Age	REC_Age	REC_Age
2	REC_TotalAvgExeTime	Duration	REC_TotalAvgExeTime
3	REC_TotalTransitionRate	REC_TotalTransitionRate	Duration
4	Duration	REC_RecentTransitionrate	EC_TotalTransitionRate
5	REC_LastExeTime	REC_RecentAvgExeTime	REC_TotalExeRate
6	REC_LastVerdict	REC_TotalExeRate	REC_LastFailureAge
7	REC_TotalFailRate	REC_TotalFailRate	TES_PRO_OwnersExperience
8	REC_LastFailureAge	REC_TotalAvgExeTime	TES_COM_CountDeclClassMethod
9	REC_RecentMaxExeTime	REC_LastFailureAge	REC_RecentMaxExeTime
10	REC_TotalExeRate	REC_RecentFailRate	TES_COM_MaxEssential

Table III shows the top 10 features identified by all three ML models using the SHAP value. These features help software testers interpret our ML model and identify which features contribute most to test case prioritization. From the table, we can see that REC_Age is the most contributing feature in all three models, while Duration is the second most contributing feature common to all three models. There are five features in total—REC_Age, REC_TotalAvgExeTime, Duration, REC_LastFailureAge, and REC_TotalExeRate—

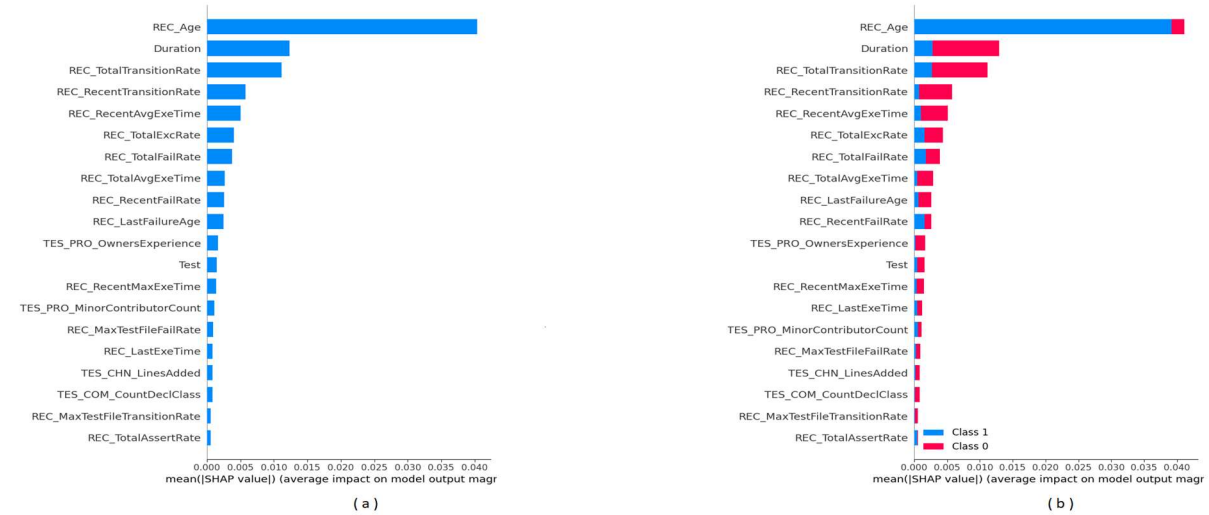

Fig. 4. Feature importance graph using SHAP for 1DCNN model outputs in TCP (a) regression-prioritized test cases (b) classification of test case verdicts

that are common among the top ten contributing features in all three models.

Figure 4 illustrates the feature importance graph for the 1DCNN model, which has dual outputs: one for regression-prioritized test cases and another for classifying test case verdicts into pass or fail. From the figure, it is clearly visible that both graphs show the same features in the same order up to the top ten features, except for REC_RecentFailRate and REC_LastFailureAge, which are in the ninth and tenth positions in graph A and the tenth and ninth positions in graph B, respectively.

TABLE IV. TAPRIORITIZED TEST CASES AND THEIR VERDICTS.

ML models	Test cases with verdicts.
1DCNN	(1942, 1), (1183, 1), (1179, 1), (1924, 1), (1905,1), (1916,1), (1085, 0)

Local-level explainability is performed to analyze any output the DL model generates in more detail. For example, Table IV displays verdicts of seven testcases with their IDs in prioritized test cases. Within this context, there are a total of four confirmed failed verdicts present in the build. The 1DCNN successfully predicts all four True Positive (TP) verdicts (1942, 1), (1183, 1), (1179, 1), (1924, 1) while concurrently making two False Positive (FP) predictions (1905, 1), (1916, 1). The tester can go through local-level explainability to understand the output generated by 1DCNN software. The tester will do so by examining three instances. The first instance showcases a TP (Figure 5-a, b), the second instance exemplifies an FP (Figure 5-c,d), and the third instance illustrates a TN (Figure 5-e,f) prediction from Table IV. Figure 5 shows two types of plots. The first is the waterfall plot, and the second is the force plot by SHAP.

The waterfall plots (Figures 5-a, 5-c, 5-e) display the impact of different features on predictions. They use colors to represent whether a feature's contribution is positive (shown in red) or negative (shown in blue), using SHAP values. SHAP values help us understand the importance of each feature. In Figure 5-a, when focusing on True Positive (TP) predictions, we can observe that the "REC_Age" feature with a value of 0.018 has the most positive impact (SHAP value of 0.02) on the prediction. Alongside this, "duration" and "REC_TotalMaxExeTime" also have positive impacts, while "REC_LastVerdict," with a value of 0.333, has the highest negative impact.

Moving to the False Positive (FP) prediction in Figure 5-c, we see that the only significant feature is "REC_LastVerdict," with a value of 1 and a SHAP value of 0.55. This feature alone is responsible for the FP prediction, while the other feature contributions are minimal.

For True Negative (TN) predictions shown in Figure 5-e, we notice that the "REC_LastVerdict" feature with a value of 0.333 has the most negative impact. The second-highest negative impact comes from the "REC_Age" feature. On the positive side, "REC_TotalMaxExeTime" and "duration" contribute positively to this prediction.

We can also gain a similar understanding from the force plots (Figures 5-b, 5-d, 5-f) for TP, FP, and TN predictions. These plots have a base value, and if the calculated SHAP value is higher than the base value, the prediction is assigned to class 1; otherwise, it's assigned to class 0. In Figure 5-b, the base value is 0.403, and the calculated SHAP value is 0.42, indicating a class 1 prediction (failure). Red indicates positive contribution, and blue indicates negative contribution. The length of the colored bands illustrates the impact. In this case, "REC_Age" has the highest positive impact, while "REC_LastVerdict" has the highest negative impact.

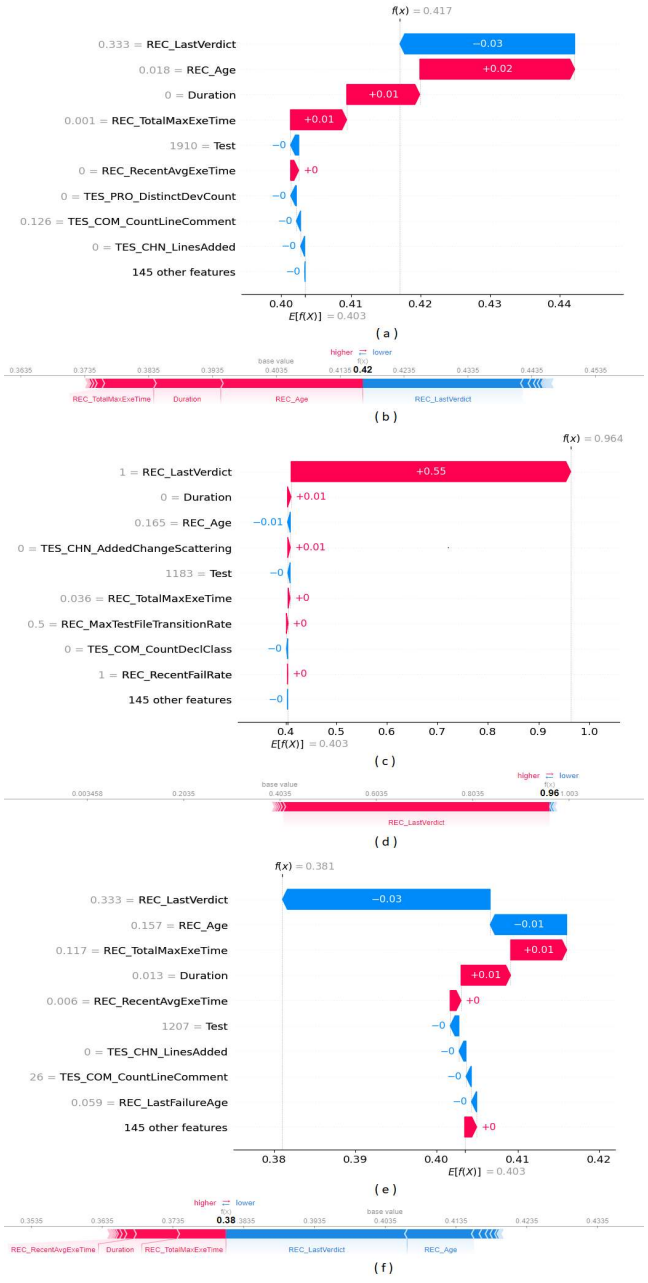

Fig. 5. Waterfall and Force plot of 1DCNN model

Figure 5-d represents an FP prediction with a base value of 0.403 and a calculated SHAP value of 0.96, which wrongly leads to a class 1 prediction. In reality, it should be class 0. The "REC_LastVerdict" is the sole feature responsible for this FP prediction.

For TN predictions shown in Figure 5-f, the base value is 0.403, and the calculated value is 0.38, making it a class 0 prediction (passed verdict). The most positively impactful feature is "REC_TotalMaxExeTime," while the most negatively impactful feature is "REC_LastVerdict."

VI. Conclusion

In conclusion, this paper tackles the challenge of efficient regression testing in CI environments by proposing an incremental learning-based ML approach for Test Case Prioritization. It evaluates three ML models—XGBoost, Logistic regression, and 1DCNN on a dataset with 150 features from 20 open-source projects. 1DCNN is identified as the top performer due to its efficiency and generalization, while XGBoost, though slightly better in RMSE and LR, performs worst. The study underscores the importance of the model explainability, highlighting that SHAP with 1DCNN effectively identifies the most contributing features, resulting in a highly interpretable and generalized TCP model.

The future scope of this approach includes incorporating more datasets to understand TCP model adaptability better, exploring a wider range of ML models and interpretation methods for more effective strategies, and integrating transfer learning into 2CNN-based TCP models to achieve improved results. Developing domain-specific features and refining explainability techniques can enhance model performance and interpretation, leading to more robust and applicable TCP models in diverse software testing environments.

References

[1] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang and Q. Wang, "Software Testing With Large Language Models: Survey, Landscape, and Vision," in IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 911-936, April 2024, doi: 10.1109/TSE.2024.3368208.

[2] Li Y, Wang Z, Wang J, Chen J, Mou R, Li G. Semantic-aware two-phase test case prioritization for continuous integration. Softw Test Verif Reliab. 2024; 34(1):e1864. https://doi.org/10.1002/stvr.1864.

[3] Karatayev, A., Ogorodova, A. and Shamoi, P., 2024. Fuzzy Inference System for Test Case Prioritization in Software Testing. arXiv preprint arXiv:2404.16395.

[4] N. Weeraddana, M. Alfadel and S. McIntosh, "Characterizing Timeout Builds in Continuous Integration," in IEEE Transactions on Software Engineering, vol. 50, no. 6, pp. 1450-1463, June 2024, doi: 10.1109/TSE.2024.3387840.

[5] Singh, A., Singhrova, A., Bhatia, R. and Rattan, D. (2023). A Systematic Literature Review on Test Case Prioritization Techniques. In Agile Software Development (eds S. Hooda, V.M. Sood, Y. Singh, S. Dalal and M. Sood). https://doi.org/10.1002/9781119896838.ch7.

[6] J. Liang, S. Elbaum, and G. Rothermel, “Redefining prioritization: continuous prioritization for continuous integration,” in Proceedings of the 40th International Conference on Software Engineering, Gothenburg Sweden: ACM, May 2018, pp. 688–698. doi: 10.1145/3180155.3180213.

[7] Wang, X., & Zhang, S. (2023). Cluster-based adaptive test case prioritization. Information and Software Technology, 165, 107339. https://doi.org/10.1016/j.infsof.2023.107339.

[8] Khan, M.A., Azim, A., Liscano, R., Smith, K., Chang, Y.K., Tauseef, Q. and Seferi, G., 2024, April. Machine Learning-based Test Case Prioritization using Hyperparameter Optimization. In Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (pp. 125-135).

[9] Marijan, D. Comparative study of machine learning test case prioritization for continuous integration testing. Software Qual J 31, 1415–1438 (2023). https://doi.org/10.1007/s11219-023-09646-0.

[10] H. Hwang and J. Shin, "Test Case Prioritization with Z-Score Based Neuron Coverage," 2024 26th International Conference on Advanced Communications Technology (ICACT), Pyeong Chang, Korea, Republic of, 2024, pp. 23-28, doi: 10.23919/ICACT60172.2024.10471933.

[11] N. Medhat, S. M. Moussa, N. L. Badr, and M. F. Tolba, “A Framework for Continuous Regression and Integration Testing in IoT Systems Based on Deep Learning and Search-Based Techniques,” IEEE Access, vol. 8, pp. 215716–215726, 2020, doi: 10.1109/ACCESS.2020.3039931.

[12] M. Hasnain, M. F. Pasha, C. H. Lim, and I. Ghan, “Recurrent Neural Network for Web Services Performance Forecasting, Ranking and Regression Testing,” in 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Lanzhou, China: IEEE, Nov. 2019, pp. 96–105. doi: 10.1109/APSIPAASC47483.2019.9023052.

[13] Z. Q. Zhou, C. Liu, T. Y. Chen, T. H. Tse, and W. Susilo, “Beating Random Test Case Prioritization,” IEEE Trans. Reliab., vol. 70, no. 2, pp. 654–675, Jun. 2021, doi: 10.1109/TR.2020.2979815.

[14] A. M. Sinaga, R. A. B. Yohana, A. Sijabat, and Y. C. Manullang, “Test Case Prioritization by Combining Mirror Adaptive Random Testing and Forgetting,” in 2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM), Laguboti, North Sumatra, Indonesia: IEEE, Oct. 2022, pp. 1–5. doi: 10.1109/ICOSNIKOM56551.2022.10034907.

[15] M. Abdelkarim and R. ElAdawi, “TCP-Net: Test Case Prioritization using End-to-End Deep Neural Networks,” in 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Valencia, Spain: IEEE, Apr. 2022, pp. 122–129. doi: 10.1109/ICSTW55395.2022.00034.

[16] A. Sharif, D. Marijan, and M. Liaaen, “DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing,” in 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), Luxembourg: IEEE, Sep. 2021, pp. 525–534. doi: 10.1109/ICSME52107.2021.00053.

[17] Z. Li and D. Hoiem, “Learning without Forgetting,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 12, pp. 2935–2947, Dec. 2018, doi: 10.1109/TPAMI.2017.2773081.

[18] Lundberg, Scott M., and Su-In Lee. "A unified approach to interpreting model predictions." Advances in neural information processing systems 30 (2017).

[19] Elbaum, Sebastian, Alexey G. Malishevsky, and Gregg Rothermel. "Test case prioritization: A family of empirical studies." IEEE transactions on software engineering 28.2 (2002): 159-182.

[20] Malishevsky, Alexey G., et al. "Cost-cognizant test case prioritization." (2006): 2006.

[21] A. S. Yaraghi, “Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts.” Jun. 16, 2023. Available: https://github.com/Ahmadreza-SY/TCP-CI