PRIORITIZING TEST CASES THROUGH FEDERATED LEARNING APPROACH

B A Sabarish, Malathi P*, C V Jai Prasanth, Kishore A V, Naveen S S, Sai Ramya Illuri

Department of Computer Science and Engineering, Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India.

Contributing authors: ba_sabarish@cb.amrita.edu, p_malathy@cb.amrita.edu, cb.en.u4cse20224@cb.students.amrita.edu, cb.en.u4cse20231@cb.students.amrita.edu, cb.en.u4cse20243@cb.students.amrita.edu, cb.en.u4cse20252@cb.students.amrita.edu.
;

Abstract

Test case prioritization (TCP) is an important part of software testing for maximizing efficiency of the resources and detecting faults as early as possible. The traditional TCP methods use historical data and predefined heuristics which are not flexible enough to adapt to dynamic software environment. In this paper we suggest a new architecture with federated learning (FL) technologies to improve TCP performance. On one hand, Federated learning allows for collaborative training of models across distributed data sources without a need for centralizing sensitive data, which in turn makes it suitable for TCP in CI software environmental setup. In this paper, we propose a comprehensive framework for integrating FL into TCP by considering factors like privacy of data and model aggregation strategy into account. Furthermore, we carry our extensive evaluation on datasets to demonstrate the efficiency of the proposed method with real data. This study proposes that federated learning for TCP can provide a considerable performance boost where 84.2 percentage of the failing test cases are identified and is expected to revolutionize software testing in the future.

1 Introduction

In the realm of software development, ensuring the quality and reliability of products is essential. Central to this endeavor is the process of testing, which serves as a cornerstone for identifying and rectifying defects before deployment. As software systems become increasingly complex and dynamic, traditional testing methodologies face significant challenges in keeping pace with evolving demands. The need for innovative approaches to testing, therefore, becomes imperative to address these challenges effectively.

Test case prioritization, the process of determining the order in which test cases should be executed, plays a pivotal role in optimizing testing resources and uncovering critical defects efficiently. Yet, conventional methods often falter in addressing the diverse needs and priorities inherent in collaborative software development environments. In dynamic team settings, where requirements evolve rapidly and diverse perspectives converge, the rigid frameworks of traditional prioritization techniques struggle to adapt, leading to suboptimal outcomes and potential delays in product delivery.

This paper suggests implementing a new framework of test cases prioritization using [17] federated learning methods as a means of addressing the challenges associated with it. Federated learning which is a decentralized mechanism with an eye on privacy preservation is the most hopeful approach in resolving the shortcomings of traditional methods. Through our suggested approach, which takes into account historical data of various teams and projects while guaranteeing privacy compliance, the efficacy and efficiency of the test case prioritization on collaborative software development environments can be increased hopefully.

Cooperative learning is an integral part of assessment prioritization, which forms a new basis for testing. This method comprises of the scheduling and execution of test scenarios while taking into account the remote observations. With this approach, resources are better allocated, defect rates are increased, and the overall product quality is improved. Demonstrating its feasibility and benefits emphasize the fact that such optimized team learning during software development life[19] cycle can give way to better software quality and faster development cycles.

2 Literature Survey

This paper focuses on the principles and the technologies of cooperative learning in test prioritisation and it shows how the old-fashioned testing methods can be bumped up. Methods of operation planning and implementation of test scenarios that will account for distributed observations will be implemented whereas the area of an allocated resource and defect detection rates will be in focus so the product quality will be elevated. Through substantiating the practicability and advantages, the most successful application of group learning in a software development life cycle can serve as a guarantee to better software quality and the shortening of the development cycles.

Rongqi Pan's study of systematic literature review (2021) is a collection of some machine learning-based methods for the case selection and prioritization workflow

(TSP). In this way, data taxonomy is presented, enriching readers in how these methodological approaches are classified.

Let consider Zhenyu Chen et al.(2021), they come out with PriorCadTest, an approach created with aim of prioritising test cases in CAD software. Their method is about the number of functional units and a trainable model for testing stage that improves the effectiveness, of which they plan to enhance the software testing processes.

DeepOrder, a Deep Neural Network (DNN)-based regression model used in continuous integration (CI) test cases prioritization is introduced (Aizaz Sharif et al., 2021). This transformation results in higher reach and concrete finding of the intensity of each test case in a time-efficient mode.

Rezwana et at. (2023) implemented TransBoost, which is a Transfer learning strategy based on tree kernels, to be used in test case prioritization . Their study evaluates transfer learning efficacy on a multitude of subjects in order to streamline tests cases in a continuous integration model.

Safa Omri and Carsten Sinz (2022) declare the problem of test case prioritization as an online learning to rank model with the benefits of reinforcement learning model. Their approach decreases test cases overhead and adapts to changes keeping the optimization of prioritization for continuous loop integration maximum.

Emyreema Ja'afar and his fellowship (2021) suggest using complexity factors-based Test Case prioritization approach, where Branch Coverage Expectation (BCE) will be employed to measure complexity. The focus of this evaluation is on the proposed approach and they conducted this using the Average Percentage of Fault Detected (APFD) metric and emphasise the value of intermediate complexity factors and other factors to achieve this test case prioritization effectively.

Jackson A. Prado Lima et al presents strategies for applying COLEMAN in the case of highly configurable software (HCS) on a continuous integration basis. The study evaluates two prioritization strategies, one being the Variant Test Set Strategy (VTS) and the other being the Whole Test Set Strategy (WST). They research about the applicability and effectiveness of these strategies in the everyday human clinical studies environments.

Muhammad Waqar and others (2021) delineate how to utilize test case prioritization techniques and they also present results they empirically obtained from the methods they applied. The research reveals that such a decision is indeed a trade-off between one approach and another, which eases the challenge to select the best solution for the case prioritization.

M.Waqar et al. (2022) build a way to prioritize test suite actions in a reinforcement learning form, also with interaction recording systems and fault seeding for proof of validity. Their thence apply the performance of the five application cases to evaluate the approach compared to the baseline which are superior.

Ali Samad et al. (2021) propose a MOPSO-based test case selection mechanism, which comprehensively consider both objective and subjective issues, including execution time, fault detection power, and coverage broadness. Their paper in turn explains the performance of the MOPSO compared one to the other criteria techniques and also it illustrates the performance of the MOPSO in different datasets.

Using a federated learning approach for test case prioritization offers significant advantages over traditional methods. Firstly, it enhances data privacy and security by allowing multiple parties to train a model collaboratively without sharing their data, thus protecting sensitive information and adhering to privacy regulations like GDPR. It is also highly scalable, leveraging data from various sources without the need for centralization, which is particularly useful for diverse teams or departments. This method reduces data transfer overhead by sharing only model updates, making it ideal for scenarios with limited network bandwidth or large datasets. Furthermore, its decentralized nature reduces the risk of a single point of failure, ensuring the learning process can continue even if one node fails. Finally, federated learning aligns with strict data governance policies by keeping data local and minimizing centralization, making it a compliant and efficient solution for organizations.

3 Methodology

3.1 Dataset Collection

Version Control System(VCS) and Continuous Integration(CI) are vital in software development, working together to enhance efficiency and quality. VCSs track code changes, ensuring stability, while CI automates builds triggered by code changes or resource availability. CI builds generate logs with details like build identifiers and test results, accessible in structured or raw text format. The dataset [5] contains the below features which are extracted from those logs.

3.2 Feature Selection and Preprocessing

According to [5], TCP considered the 'last execution time' or 'average test execution time' feature for model training and achieved good results. The hypothesis behind considering this feature is that the test suite with higher execution time may cover more significant parts of the code and thus may contain more faults. However, many other factors can influence the test execution time, such as machine speed. In large-scale CI projects, the test runs parallel on different machines, which do not necessarily have the same configuration. A test suite run on a low-configuration machine can have a higher execution time which contradicts the above hypothesis. Therefore, this

Table 1 Description of Features

Feature	Description
run_id	Unique identifier for each test run.
stage_id	Identifier for the stage of testing.
branch	Branch of code being tested.
run_timestamp	Timestamp for when the test run occurred.
stage_timestamp	Timestamp for when the testing stage occurred.
stage_duration	Duration of the testing stage.
test_suite_duration	Duration of the test suite execution.
Failure_Count	Number of failures encountered by the test.
Transition_Count	Number of transitions between test states.
Last_Failure_Age	Age of the last failure encountered by the test.
Last_Transition	Age of the last state transition.
avg_duration	Average duration of test execution.
has_transitioned	Flag indicating if the test has transitioned states.
Failed	Flag indicating if the test case has failed.
commits_len	Number of commits associated with the test.
#Files_Changed	Number of files changed by the test.
authors_distinct_len	Number of distinct authors associated with the test.
file_ext_distinct_len	Number of distinct file extensions affected by the test.
min_filepath_distance_lv	Minimum file path distance.
Match_Count	Number of matches found.
min_filename_distance_lv	Minimum filename distance.
max_freq	Maximum frequency.
max_freq_rel	Maximum frequency relative.
max_failure_freq	Maximum failure frequency.
max_failure_freq_rel	Maximum failure frequency relative.
last_test_suite_duration	Duration of the last test suite.
Test_Total	Total number of tests.
#Lines_Inserted	Number of lines inserted by the test.
#Lines_Deleted	Number of lines deleted by the test.
failed_before	Flag indicating if the test failed previously.
Execution_Count	Number of times the test has been executed.
Failure_Rate	Rate of test failures.
Last_Failure	Timestamp of the last failure encountered by the test.
Test_Identifier_string	Identifier string for the test.

paper does not consider any features that are solely dependent on the test execution time and selects the following features for the TCP.

CI Features:

- Test_Identifier
- Failure_Count
- Last_Failure_Age
- Last_Failure
- Failure_Rate
- Transition_Count
- Last_Transition

VCS Features:

- #Files_Changed

- #Lines_Inserted
- #Lines_Deleted

Preprocessed the datasets to handle missing values and encode the 'Failed' column using label encoding to convert it into a numerical format. 1 represents True (test case has failed) and 0 represents False (test case has not failed). To address class imbalance issues in the dataset we used SMOTE (Synthetic Minority Over-sampling Technique) which generates synthetic samples for the minority class (Failed=1) to address class imbalance by oversampling.

3.3 FL Architecture

Federated learning is considered a cutting edge training approach wherein a centralized server collaborates to build a shared global model. What stands out about it is its clever way of handling data privacy — all sensitive information resides safely either in the identity institutions or on individual devices, where the data originates. By enabling collaboration without the need for centralized data storage, federated learning redefines how machine learning models are trained, offering a robust solution for connecting fragmented data sources in a manner that prioritizes and preserves privacy.

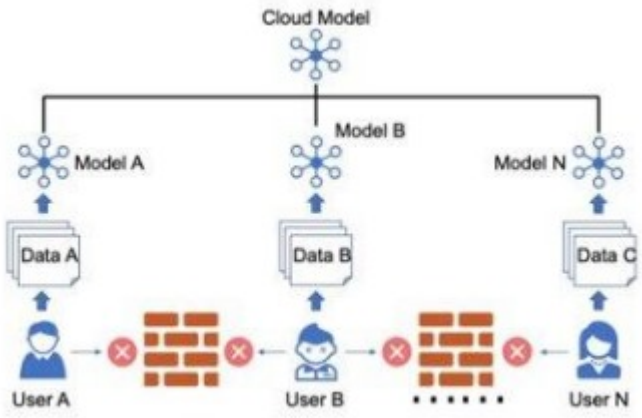

Fig. 1 FL Architecture. Source: Federated Learning with Adaptive Batchnorm for Personalized Healthcare (2021)

Tensor Flow Federated (TFF) is an open source yet a power tool which is instrumental to conform AI models to the agility level of partner-oriented software development. It also offers training models to not be available at shared devices among which their data wouldn't be shared by collaborators as well, hence addressing privacy concerns. TFF can help us train even the teams we do not supervise while guaranteeing that the data's security is preserved completely. With the ability to verify and to estimate the federated machine learning, the implementation will be the greatest factor to refine our solution before applying them in the actual implementation.

Here, every client has a neural network model working by using a binary cross-entropy loss function and binary accuracy metrics, and with a sigmoid activation function that has been fit to work in a binary classification problem. This architecture is tailored for the prediction of test case outcomes where 1 means that a test case will fail and the value 0 indicates that it will pass. The output range of the sigmoid function from 0 to 1 serves as a useful basis for the test case prioritization where the failure probability is the leading factor.

3.4 Training the Federated Learning Model

Federated averaging is a technique used to train machine learning models where data is spread across many different clients. In the conventional approach of centralized machine learning, data is collected and stored in a central server, and a single model is trained on this consolidated data. Following with opposite direction in federated averaging, data stays decentralized, having many places like devices and edge devices, and model is trained locally on every device with specific condition. Potentuine have a selection of customers which participate in each training cycle. These clients receive the global model, and a training on their own specific datasets is conducted by them. Next, averaging method is applied to all client-side models' parameters which are then sent to the main server for centralized learning. This amassing technique that employs knowledge on various temporal clients offered as input to the global model while privacy remains the main concern. In the process of training error is computed in multiple rounds and the process repeats until convergence or the desired level of performance is achieved. This methodological process is effective thanks to collaboration and the iterative nature of the FedAvg algorithm which ensure that data from various geographical locations goes through a privacy protection process during model training.

3.5 Workflow Mechanism

The process of federated averaging involves the following steps:

1. **Initialization:**
   - The central server initializes and creates a global model.
2. **Client Selection:**
   - A set of clients is chosen to participate in the training round.
   - Client selection may be arbitrary or determined by predefined standards.
3. **Model Distribution:**
   - The server sends the chosen set of clients receive a copy of the global model.
   - Each client gets an exact duplicate of the model.
4. **Local Training:**
   - The Clients train the model using their local data.

- The training process on each client systems may involve multiple iterations or epochs to improve the model's performance.

5. **Model Aggregation:**
   - After local training, updated models from all clients are sent back to the central server.
6. **Model Averaging:**
   - The central server aggregates the received models by averaging their parameters.
   - This averaging process ensures that the global model benefits from the knowledge learned across different clients while preserving privacy.
7. **Repeat:**
   - Steps 2 through 6 are repeated for multiple training rounds.
   - The process continues until convergence or the desired level of performance is achieved.

3.6 Evaluation of the Federated Learning Model

In the context of test failure prediction, most real-world datasets are imbalanced, with the number of failed tests being relatively small compared to the number of successful tests. Therefore, accuracy alone is not an efficient way to measure the performance of a predictive model. Instead, this paper focuses on Recall and F-measure metrics for evaluation.

Metrics Calculation

To calculate precision, recall, and F-measure, it is necessary to identify the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) from the prediction results.

- **TP (True Positive):** Indicates the number of failed tests correctly identified.
- **TN (True Negative):** Represents the number of passed tests correctly identified.
- **FP (False Positive):** Points out the number of passed tests misclassified as failed tests.
- **FN (False Negative):** Indicates the number of failed tests misclassified as passed.

The metrics are calculated as follows:

- **Precision:** precision $= \frac{TP}{TP+FP}$
- **Recall:** recall $= \frac{TP}{TP+FN}$
- **F-measure:** F-measure $= \frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$

3.7 Class Imbalance Methods

We implemented the following class imbalance methods and evaluated the model:

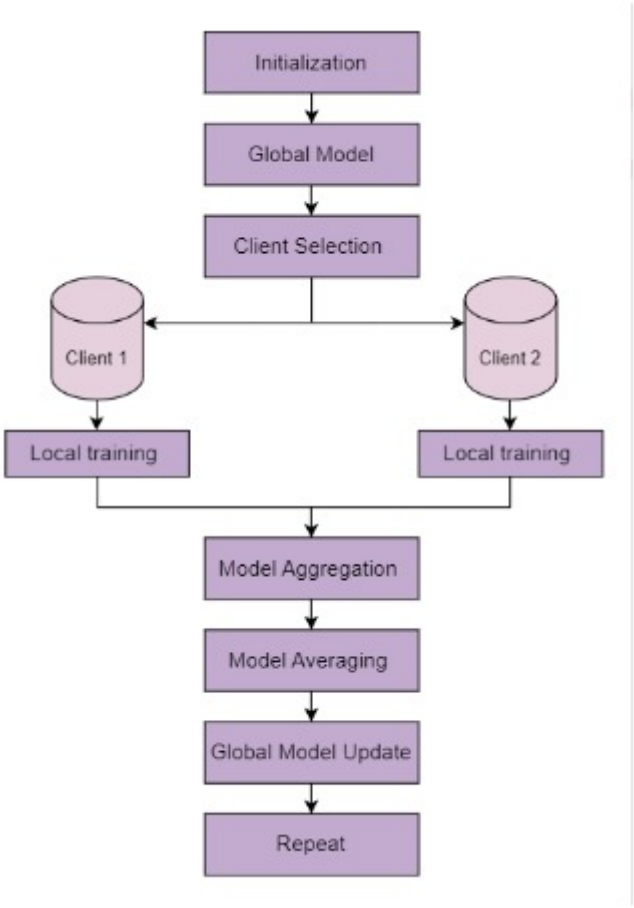

Fig. 2 Work Flow

3.7.1 Class Weights

Initially, class weights were incorporated into the model training process to account for the disproportionate representation of passing and failing test cases. By assigning higher weights to the minority class (passing test cases), the model aimed to learn more effectively from these instances. However, while achieving an impressive overall accuracy of 90%, closer examination revealed a bias towards classifying most test cases as failing, highlighting the limitations of this approach in effectively capturing the nuances of the dataset.

3.7.2 Upsampling

Recognizing the need for a more balanced representation of both classes, the upsampling technique was employed. This involved augmenting the dataset by duplicating instances of passing test cases, thereby equalizing the class distribution and providing

	Predicted	
Actual	42448	198
	548	60

Confusion Matrix For CW

the model with a more comprehensive set of examples to learn from. The implementation of upsampling yielded promising results, with a subsequent accuracy of 70% and a notable improvement in the classification matrix, indicating a more equitable classification of passing and failing test cases.

	Predicted	
Actual	35484	5222
	16892	28365

Confusion Matrix For UpSampling

3.7.3 SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE was introduced as an interesting development on upsampling concept which is meant to overcome class imbalance problem. Unlike the traditional upsampling, which only straightly replicates the existing ones, SMOTE finds the new ones by interpolating between neighboring minority class instances. This consequently leads to high-quality and wider-spanning dataset to use in the learning process, which improves the resilience of the model. SMOTE was one of the key factors that contributed to the refinement of the model. The model accuracy was 80% after SMOTE application, showing the accuracy of the class balance and the effectiveness of the classification.

	Predicted	
Actual	38653	4053
	9082	33624

Confusion Matrix For SMOTE

3.8 Prioritization

Test failure prediction involves the proactive identification of tests likely to fail before they are executed. By analyzing historical test execution data, predictive models can be built to forecast which tests are prone to failure based on various factors such as code changes since the last execution and the tests' execution history. The goal is to anticipate problematic tests early in the testing process, thereby saving time and resources. By identifying tests that are likely to fail in advance, developers and testers can prioritize fixing them before execution, ultimately reducing the overall test execution time[18,19].

Test case prioritization aims to ensure that the most critical tests are executed first, thereby maximizing defect detection and overall software quality. Rothermel et al. provided a formal definition of the TCP problem, where for a given test suite T, a set of permutations of T as PT, and a scoring function f, the objective is to find a prioritization T' in PT such that its score $f(T')$ is greater than or equal to the score of any other prioritization in PT. In simpler terms, TCP techniques ensure that tests are ordered based on their importance, considering factors such as the likelihood of failure, criticality of the tested code, and the cost of fixing defects.

$$T' \in PT \text{ such that } (\forall T'')(T' \in PT)(T' \neq T'')[f(T') \geq f(T'')]$$

4 Results

The federated learning approach successfully trained binary classification models on multiple datasets containing test case execution records. Evaluation metrics such as accuracy, precision, recall, and F1-score demonstrate the effectiveness of the test case prioritization strategy. The federated learning approach shows promise in improving defect detection rates and optimizing resource utilization compared to traditional methods.

The evaluation of the test case prioritization model encompassed a comprehensive analysis of its performance using various metrics and techniques:

- **Accuracy:** The accuracy metric is the fundamental measure of how much the model's overall correctness in prioritizing test cases. Initial attempts utilizing class weights resulted in an accuracy of 85% and Upsampling improved the accuracy to 75%, and SMOTE further enhanced it to 84%.

- **Precision, Recall, and F1-Score:** Precision indicates the instances among all the instances that are predicted as positive which were correctly identified. Recall is the proportion of the true positives that were correctly identified. F1-score provides a balance between precision and recall but it is a bit more complicated to calculate. Upsampling and SMOTE methods most often enhanced precision, recall and F1-score while the class weights were used by themselves.

Technique	Accuracy	Precision	Recall	F1 score
Class Weight	93.36%	0.03	0.01	0.01
Upsampling	77.49%	0.81	0.68	0.75
SMOTE	84.28%	0.87	0.76	0.83

Table 2 Performance Metrics for Different Techniques

5 Conclusion

This research attempts the identify of the new techniques in test case prioritization in the area of the collaborative software development using the federated learning techniques. The outcomes hence demonstrate that federated learning has an anniversary of privacy-preserving and scalable issues in the test case prioritization based on historical test execution date TFF, which is the TensorFlow Federated framework, can be considered as a powerful option that enables the implementation of decentralized algorithms in the field of federated learning while disregarding privacy issues and determining scale risk. Next step could be devotion to the perfecting the federated learning algorithm, the investigation of more precise performance measures, and actual deployment to check whether the method we proposed can work everywhere, or it only does well in motivated situations. .

References

1. Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., ... Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

2. Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering*, 27(2), 29.

3. Zeng, F., Liu, S., Yang, F., Xu, Y., Zhou, G., & Xuan, J. (2022). Learning to Prioritize Test Cases for Computer Aided Design Software via Quantifying Functional Units. *Applied Sciences*, 12(20), 10414.

4. Sharif, A., Marijan, D., & Liaaen, M. (2021, September). DeepOrder: Deep learning for test case prioritization in continuous integration testing. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 525-534). IEEE.

5. Mamata, R., Azim, A., Liscano, R., Smith, K., Chang, Y. K., Seferi, G., & Tauseef, Q. (2023, May). Test Case Prioritization using Transfer Learning in Continuous Integration Environments. In *2023 IEEE/ACM International Conference on Automation of Software Test (AST)* (pp. 191-200). IEEE.

6. Lima, J. A. P., Mendonça, W. D., Vergilio, S. R., & Assunção, W. K. (2020, October). Learning-based prioritization of test cases in continuous integration of highly configurable software. In *Proceedings of the 24th ACM conference on systems and software product line: Volume A-Volume A* (pp. 1-11).

7. Ja'afara, E., Sa'adah Hassana, S. B., & Ahmada, J. (2018). Test Case Prioritization Approach for Sequence of Events Using Complexity Factor.

8. Springer. (2021). Test Case Prioritization Approach. *Empirical Software Engineering*, 27(2), 29. Retrieved from https://link.springer.com/article/10.1007/s10664-021-10066-6

9. BrowserStack. (n.d.). Test Case Prioritization. Retrieved from https://www.browserstack.com/guide/test-case-prioritization

10. Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017, July). Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 12-22).

11. Samad, A., Mahdin, H. B., Kazmi, R., Ibrahim, R., & Baharum, Z. (2021). Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method. *Scientific Programming*, 2021, 1-13.

12. Waqar, M., Imran, Zaman, M. A., Muzammal, M., & Kim, J. (2022). Test suite prioritization based on optimization approach using reinforcement learning. *Applied Sciences*, 12(13), 6772.

13. Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (1999, August). Test case prioritization: An empirical study. In *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)* (pp. 179-188). IEEE.

14. Omri, S., & Sinz, C. (2022, May). Learning to rank for test case prioritization. In *Proceedings of the 15th Workshop on Search-Based Software Testing* (pp. 16-24).

15. Retrieved from https://figshare.com/s/851513ca9907a10dd17d

16. Rajagopal, Shinu M., M. Supriya, and Rajkumar Buyya. "FedSDM: Federated learning based smart decision-making module for ECG data in IoT integrated Edge-Fog-Cloud computing environments." Internet of Things (2023): 100784 .

17. Bisht, N.S., Duttagupta, S. "Deploying a Federated Learning Based AI Solution in a Hierarchical Edge Architecture", IEEE Region 10 Humanitarian Technology

Conference, R10-HTC, 2022, 2022-September, pp. 247 - 252, DOI: 10.1109/R10-HTC54060.2022.9929526

18. Sabarish B A , Arunkumar Chinnaswamy . User Story based Automatic Test Case Classification and prioritization using NLP (Natural Language Processing) based Deep Learning. *TechRxiv.* September 27, 2023.

19. Jaagruthi, A., Varshitha, M., Vinaya, K.S., Gupta, V.N., Arunkumar, C., Sabarish, B.A. (2023). User Story-Based Automatic Keyword Extraction Using Algorithms and Analysis. In: Bhateja, V., Carroll, F., Tavares, J.M.R.S., Sengar, S.S., Peer, P. (eds) Intelligent Data Engineering and Analytics. FICTA 2023. Smart Innovation, Systems and Technologies, vol 371. Springer, Singapore. https://doi.org/10.1007/978-981-99-6706-3_30