Minimax Theory and its Applications

Volume 11 (2026), No. 3s, 123 to 136

# Digital Twin-Driven Adaptive Graph Learning for Explainable Software Reliability Prediction

**M.Jeevana Sujitha[1], Dr.D.Naga Malleswari[2], Kodukula Subrahmanyam[3]**

[1]Research scholar, Department of Computer Science and Engineering, KoneruLakshmaiah Education Foundation,Vaddeswaram,AP, India, Email: jeevana.srkrcse@gmail.com
[2]Associate Professor, Department of Computer Science & Engineering, KoneruLakshmaiah Education Foundation, GreenFileds, Vaddeswaram, A.P., Email: nagamalleswary@gmail.com
[3]Professor, Department of CSE, Dr.RVR NRI Institute of Technology Deemed to be University, India Email: smkodukula@yahoo.com

**Abstract:** Software reliability prediction is a fundamental activity in software engineering because unreliable software can lead to significant operational, financial, and safety-related consequences. However, existing reliability prediction approaches predominantly rely on static historical datasets and offline analysis, thereby limiting their ability to capture the dynamic behaviour of modern software systems operating under continuously evolving workloads and execution environments. To address these limitations, this paper proposes a novel Digital Twin-Driven Adaptive Graph Learning framework for Explainable Software Reliability Prediction (DTAGL-ESRP). The proposed framework constructs a software digital twin that continuously synchronizes runtime execution traces, operational profiles, software logs, and structural code metrics to create a dynamic virtual representation of the software system. Subsequently, a Temporal Graph Transformer is employed to model inter-module dependencies and capture fault propagation patterns across software components. To maintain prediction robustness under evolving operational conditions, a concept drift detection mechanism is integrated to automatically adapt the learning process whenever significant behavioural changes are observed. Additionally, an explainable artificial intelligence module based on SHAP analysis is incorporated to identify critical reliability-influencing attributes and provide transparent reliability assessment. Experimental evaluation conducted on benchmark NASA dataset demonstrates that the proposed DTAGL-ESRP framework significantly outperforms conventional reliability prediction models in terms of prediction accuracy, adaptability, and interpretability.

Keywords: DigitalTwin, Software Reliability Prediction, Temporal Graph Transformer, Adaptive Graph Learning, Concept Drift Detection, Shap-Based Explainability, Software Dependency Graph.

## Introduction

In today's software-dominated arena of healthcare, aerospace, financial, industrial automation and cloud computing, the reliability of software is an important quality attribute that can lead to significant economic losses, disruption of service and safety concerns when the software fails. Software systems change often, with dynamic workloads and constantly changing environments, it is hard to accurately predict software reliability. If a software module is identified early as potentially unreliable, the software developer may be able to optimize the software testing by creating a more efficient testing plan.If a software module is determined to be potentially unreliable, the software developer should consider optimizing the software testing by designing a more efficient testing plan, which can help avoid failures after the software is deployed. As a result, software prediction techniques that are reliable are essential for enhancing software quality assurance and for enabling proactive maintenance in the software development lifecycle [1].

Although there has been great progress in the field of software reliability prediction, the problem is still technically challenging because of the heterogeneous nature of software engineering data and the constantly changing behavior of software systems. Reliability assessment is based on several information sources – software metrics, execution logs, operational profiles, exception reports, dependency structures, etc. – each of which provide some information about software behavior. The sources are heterogeneous, and frequently characterized by noise, high dimensionality, and low fault occurrence and complex inter-modular interactions that vary over time [2]. Furthermore, software concept drift arises from changing statistical properties of software behavior due to software changes such as software updates, software configuration modifications, and changes in the workloads, which makes models trained from static historical datasets less effective. These

difficulties make it an unresolved research question to have accurate and adaptive software reliability prediction [3].

Current software reliability prediction techniques include statistical, traditional machine learning, ensemble learning, deep neural networks, graph neural networks, and Transformer-based architectures. Handcrafted software metrics are the basis of statistical and conventional machine learning approaches, and many of them tend to assume that the data distribution is stationary, making them less flexible and adaptable to varying execution conditions. Deep learning models enhance feature representation, yet they tend to deal with software modules individually without consideration for structural dependency relationships leading to fault propagation. While graph-based techniques are better suited to represent software dependency, most of the available models are static, and do not model temporal evolution. Likewise, Transformer models offer better long range dependency learning but typically do not have continuously synced runtime models, which means they are less useful in real-time software reliability management [4].

The current architectures are fundamentally constrained by their inability to keep stable representations of their software while continuously changing execution environments. Representation drift, contextual inconsistency and weakened prediction reliability are all caused by static feature representations that become increasingly outdated with changes in software behaviour. Additionally, because of poor preservation of dependency evolution between modules, exact modelling of dynamic fault propagation is limited to software components. Current techniques are also not very interpretable, and it is difficult to see how to determine from the available software metrics which are responsible for the predicted reliability degradation. This is why software engineers often do not have a clear and flexible decision support system that can model the changing software interdependencies, identify behavioural drift and explain software predictions on the fly [5].

In pursuit of overcoming these limitations, this paper presents a proposed framework for Explainable Software Reliability Prediction (DTAGL-ESRP) based on the Digital Twin and Adaptive Graph Learning (DTAGL). The framework proposed here builds a software digital twin, which is continuously synchronized, to represent the physical software system as a virtual representation that is continually updated during runtime with execution traces, software metrics, operational profiles, exception reports, and dependency information. A Temporal Graph Transformer is trained with evolving inter-module dependency patterns and fault propagation behaviour, and an adaptive concept drift detection mechanism adds to the prediction model based on the changing software characteristics. Last but not least, a SHAP-based explainability module quantifies the contributions of individual software metrics, enabling a transparent evaluation of the reliability of the software and giving direct information for software maintenance. The key novelty and contributions of the presented work are:

(i) A novel digital twin-driven adaptive graph learning architecture that is capable of continuously adapting the software at run-time to avoid representation obsolescence;
(ii) A Temporal Graph Transformer that preserves evolving structural dependencies and fault propagation patterns for stable representation learning of software;
(iii) An adaptive concept drift learning mechanism that ensures prediction robustness in dynamic execution environments;
(iv) An explainable prediction framework based on SHAP analysis to enhance the transparency of the predictions, identifying the software metrics that influence the prediction reliability;

In this paper, we proceed as follows: The existing work is reviewed in detail in Section 2, where the drawbacks of existing techniques are pointed out and gaps in the research are identified which inspire this work. Section 3 presents a proposed framework, and a detailed explanation of the architectural components. The experimental methodology, evaluation metrics, comparative performance analysis, ablation experiments, explainability assessment and a detailed discussion on the results are presented in Section 4. In the end, Section 5 summarises the overall impact of the study, presents the main findings and suggests possibilities for further research.

**Literature Survey**

Software reliability prediction has been a great interest because of its importance in software quality enhancement, low maintenance costs and reliability software deployment. Machine learning and deep learning techniques have also recently been developed that are capable of appropriately modeling complex software behaviors. However, the poor adaptability to changing software systems, the lack of exploitation of software dependency information, limited interpretability and data imbalance remain a challenge to prediction performance.

Getachew et al. identified an ensemble learning framework for software reliability prediction which enhances the robustness of prediction by combining several machine learning models. The framework uses bagging, boosting and stacking techniques for classification and reliable estimation. The concept of Mean Time Between Failures (MTBF) is used for the estimation of software failure time, according to Musa's software reliability model; and the NASA benchmark datasets are used for software defect classification. Experimental validation found that ensemble learning can boost the accuracy of prediction to ~94% and accuracy of defects classification to ~97% as compared with individual classifiers. The proposed method, however, is largely based on handcrafted software metrics and cannot model complex structural relationships between software modules that can hinder its use in large and evolving software systems [6].

Misbah et al. proposed a smart fused software defect prediction model to predict the faulty software modules more accurately. They have two stages in which their framework is operating in two sequential stages. First, four supervised learning algorithms (Random Forest, Support Vector Machine, Naïve Bayes, Artificial Neural Networks) are optimized and trained independently. Then, a weighted voting ensemble is used as an aggregation of their classification results to obtain the final defect classification. Seven NASA MDP benchmark data sets were used to validate the model. The results of the experiments showed that the developed ensemble method outperformed many of the defect prediction methods already used in machine learning and ensemble method in all the cases. Although the framework achieves high predictive accuracy, it mainly concentrates on aggregation of classifiers without considering the information about temporal software evolution and inter-module dependency information; thus it has limited ability to model the dynamic software behavior [7].

In a hybrid approach, Ahmed Abdu et al. have developed a defect prediction model using a combination of semantic representations of source code and classical software metrics. They suggest a CNN-MLP architecture that uses Abstract Syntax Trees (ASTs) to extract semantic information and represents them as a vector using Word2Vec. The semantic features are handled by a CNN while the traditional software metrics are learned by a MLP. Finally, the results of both networks are combined and input to fully connected layers for the final prediction of defects. Various experiments conducted on several open-source software projects showed that the prediction performance was better than the performance of several effort-aware and non-effort-aware defect prediction approaches. While the hybrid architecture is able to integrate heterogeneous feature representations, it does not explicitly represent software dependency graphs or decision mechanisms that can be explained and used to support practical software maintenance [8].

Iqra et al. explored the impact of feature selection on software defect prediction in five publicly available datasets from NASA: CM1, JM1, KC1, KC2 and PC1. They used a number of classifiers from the machine learning family such as Random Forest, Logistic Regression, Multilayer Perceptron, Bayesian Networks, J48, and Support Vector Machine as well as Decision Stump, Lazy IBK and ZeroR in combination with a variety of feature selection techniques. All experiments were performed in the WEKA machine learning environment and statistical validation was done in the Minitab software package. The experimental analysis showed that the use of feature selection resulted in better prediction accuracy than the models without feature selection. However, the proposed framework is very limited in its capacity of automatically learning complex software representations in ever-changing development environments [9] and relies heavily on handcrafted feature subsets, and the use of traditional classification algorithms.

To solve the class imbalance problem, Nasraldeen and Károly introduced a software defect prediction framework that uses Bidirectional Long Short-Term Memory (Bi-LSTM) networks with oversampling. The model was tested with the benchmark datasets retrieved from the PROMISE repository with both original data and Random Oversampling, Synthetic Minority Oversampling Technique (SMOTE). The evaluation metrics were accuracy, precision, recall, F-measure, Matthews Correlation Coefficient (MCC), Area Under the ROC Curve (AUC), Area Under the Precision-Recall Curve (AUCPR) and Mean Squared Error (MSE). The experiment result indicates that the balanced datasets significantly enhance the prediction performance, with average accuracies of 94% and 92% obtained with the Random Oversampling and SMOTE, respectively, while the average accuracy for the original datasets is 88%. The average of the F-measure also rose, from 51% to 94% and 92% respectively, showing the effectiveness of the Bi-LSTM and oversampling strategies [10]

Despite the success of various software reliability and software defect prediction techniques based on ensemble learning, hybrid deep learning, feature selection and sequence-based models, most of them are based on static historical datasets and offline learning, which has restricted their use in dynamic software environment. Most approaches cannot deal with the dynamic dependency between modules, the runtime execution pattern, and the change of concepts due to software evolution and shifting workloads. Also, the lack of model interpretability makes it difficult for developers to know what is causing reliability issues. Hence, a need for a holistic framework with continuous synchronisation of the runtime software behaviour using a digital twin, modelling of

dynamic software dependencies using adaptive graph learning, automatic adaptation to concept drift, and transparent reliability predictions using explainable artificial intelligence. The proposed DTAGL-ESRP framework aims to overcome these challenges by combining the concepts of a Digital Twin, a Temporal Graph Transformer, the adaptive concept drift detection approach and explainability through SHAP into a single end-to-end framework.

**Proposed Method**

The proposed DTAGL-ESRP framework combines the Digital Twin technology with Temporal Graph Transformer-based adaptive graph learning, concept drift detection, and explainable artificial intelligence (XAI), as a single end-to-end framework that is able to continuously monitor, learn, adapt and explain software reliability predictions.The Workflow starts with the continuous data acquisition during runtime from various heterogeneous software sources. While static software metrics are used, the framework also gathers other metrics such as structure of the code, execution traces, operational profiles, runtime logs, exception reports, resource utilization statistics, and software dependency information that are generated during system operation. These data streams are heterogeneous, and describe both the structural properties and operational behaviour of software modules. The data collected are then subjected to complete pre-processing: the missing values are imputed, duplicated values are removed, the data is Z-score normalized, outliers are treated using the Interquartile Range (IQR), features are selected based on their correlation and the Synthetic Minority Over-sampling Technique (SMOTE) is used to balance the classes. These are used to enhance data quality, remove data redundancy, normalize the data features, and overcome the class imbalance problem found in software defect datasets. The high quality feature set created becomes the input for the next step of digital twin modelling. A software Digital Twin is then built to ensure that the virtual representation of the physical software system is always kept up-to-date. Unlike static software models, the digital twin continuously changes its state as new run-time data arrives. Virtual entities are used to represent each software module and include software metrics, execution statistics, dependency information, operation states, etc. and synchronization mechanisms continuously keep the virtual software model in sync with the actual system. The real-time synchronization allows the framework to pick up behavioural changes caused by software changes, deployment updates, load changes, etc. at runtime. Thus, the Digital Twin can be used to keep a precise representation of software behavior, which is continuously updated, to base adaptive reliability prediction.The software system is synchronized, and the interactions between the software modules are represented as a dynamic dependency graph. The graph nodes represent software modules and the graph edges represent inheritance relationships, function calls, package dependencies, data flows and runtime interactions. In contrast to traditional feature-vector representations, the graph structure explicitly retains dependency relationships that cause fault propagation in a system of interdependent software elements. A Temporal Graph Transformer (TGT) is then used to learn the structural dependency and temporal execution patterns simultaneously. To capture the relative importance of neighbouring software modules, multi-head self-attention mechanisms were adopted, and to account for the evolution of the behaviour over time, temporal positional encoding was used. The creation of the graph embedding consequently captured the dynamic software interactions well, and enhanced the framework's capability of predicting reliability degradation in varying execution environments.It is clear that the behaviour of software can change continuously during deployment, hence the proposed framework embraces an adaptive concept drift detection system to retain prediction robustness. The errors predicted at the online monitor are continually evaluated to look for statistically significant deviations from the historical behaviour. Whenever concept drift is detected, the learning model will be updated incrementally with new data gathered during execution and the framework will learn to adapt to changes in software attributes without having to re-train.Lastly, the learnt graph representations are delivered to a fully connected classification layer, which guesses the dependability standing of each software module. The classifier is used to estimate the likelihood of a software component being reliable or defective while running under the operating conditions at hand. A SHAP (SHapley Additive exPlanations) module to provide explainability is added to the prediction framework to enhance transparency and user confidence. For each of the software metrics, SHAP is able to calculate so-called feature contribution scores, which help software engineers pinpoint the most predictive factors. Figure 1 shows the block diagram of the proposed method.

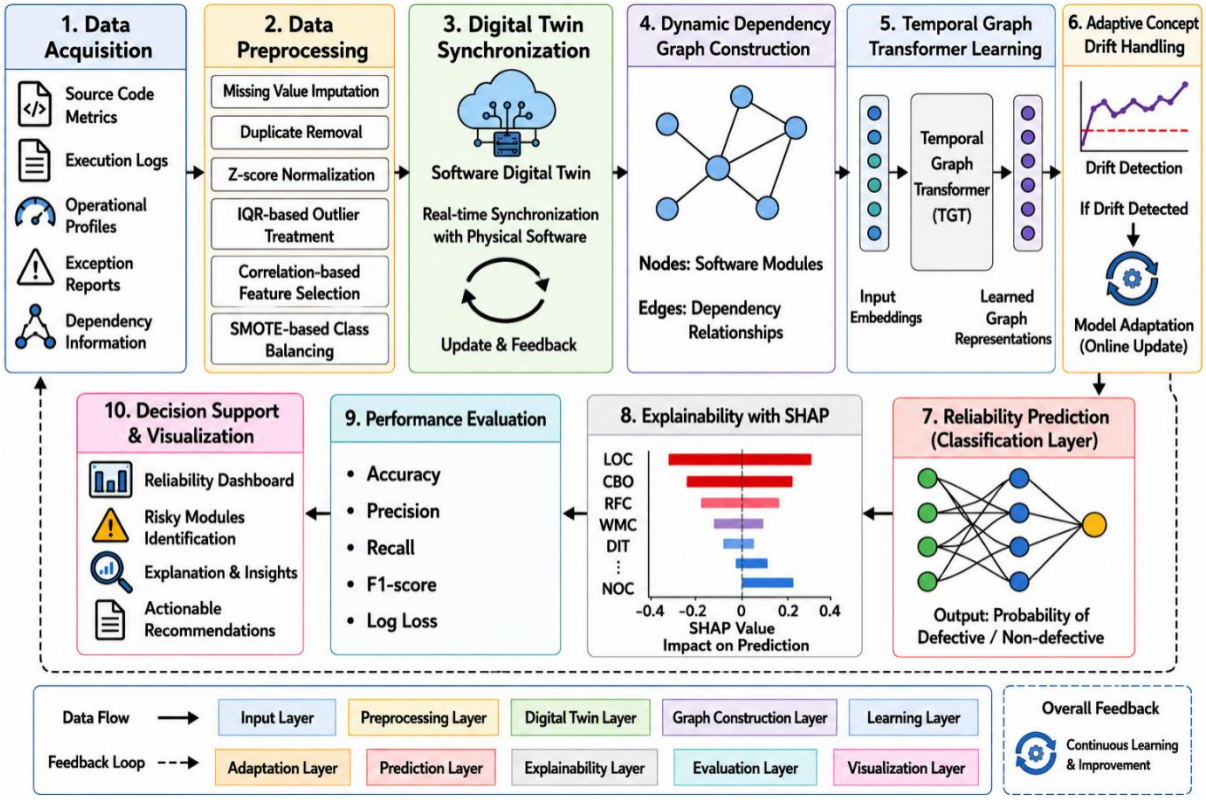



**Figure 1.** Block diagram of the proposed method

**Data Pre-processing**

A software reliability prediction model's performance is heavily reliant on the quality of the input data [11]. In order to train the proposed DTAGL-ESRP framework, both NASA datasetis consequently pre-processed through an exhaustive pre-processing pipeline. The pre-processing stage deals with data consistency, reduce noise, balance class sizes and create appropriate graph representations for learning on temporal graphs..

**a. Data Cleaning**

The raw software metric data is first cleaned for the data quality improvement, to delete missing values and duplicate records [12-13]. Numerical values that are missing are replaced by the median value and software modules that are duplicated are removed to avoid redundant information affecting learning. This pre-processing step is to ensure that the dataset is complete, consistent and good for the next analysis step is presented in Eq. (1).

$$x_i = \begin{cases} x_i & \text{if } x_i \text{ is available} \\ \text{Median(X)}, & \text{otherwise} \end{cases} \quad (1)$$

where$x_i$represents an individual feature value and X denotes all observed values of that feature.

**b. Z-Score Normalization**

To remove the differences in feature scales, all the feature metrics for software are normalized using the Z-score normalization technique [14-15]. This transformation puts each feature at zero and scales them to have unit variance, so that attributes with larger numerical ranges do not dominate the learning process. Therefore, the optimization process converges faster and have better prediction performance is presented in Eq. (2).

$$Z = \frac{X-\mu}{\sigma} \quad (2)$$

**c. IQR-Based Outlier Detection and Treatment**

The outliers are detected by windows based upon the Interquartile Range (IQR) of data between the first and third quartiles [16]. Samples outside the acceptable range are regarded as abnormal and clipped to the nearest acceptable range to minimize the effect of the samples but maintain the valuable information. This procedure increases the strength of the reliability prediction model. The IQR is calculated in Eq. (3) - Eq. (4):

$$IQR = Q_3 - Q_1 \quad (3)$$

If the data value lies outside the range of.

$$x < Q_1 - 1.5(\text{IQR}) \text{ or } x > Q_3 + 1.5(\text{IQR}) \quad (4)$$

The outliers detected are clipped or removed for improving data quality.

**d. Correlation-Based Feature Selection**

To find out the highly correlated and redundant metrics for software before model training, correlation analysis is used [17]. To eliminate the redundancy, computational complexity and model generalization, features with high mutual correlation are removed. The correlation coefficient that is calculated using Pearson's correlation for two features is shown in Eq. (5).

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2} \quad (5)$$

**e. SMOTE-Based Class Balancing**

The software defect data sets are imbalanced and they have a very low number of defective modules compared to non-defective modules [18]. To address this problem, the Synthetic Minority Over-Sampling Technique (SMOTE) is used to create synthetic minority class instances which are interpolated between the neighbouring defective instances. This balancing technique improves the classifier's performance in detecting faulty software components and reduces the prediction bias as mentioned in Eq. (6).

$$x_{new} = x_i + \lambda(x_{NN} - x_i), \quad 0 \leq \lambda \leq 1 \quad (6)$$

where$x_i$ is a minority-class sample, $x_{NN}$ is one of its nearest minority neighbors, and λ is a randomly generated interpolation factor.

**Digital Twin Synchronization**

The Digital Twin synchronization module keeps the physical software system and the virtual model consistent throughout the observation of the physical system by adding the newly observed runtime information to the digital twin model. The proposed approach for synchronizing the virtual software representation to the running program differs from the traditional approaches to software reliability prediction that are based on static snapshots of the software's history. The continuous synchronization allows the digital twin to capture the state of execution of the software, which means that all the latest characteristics of the software will be captured and the digital twin will be used to make it reliable. Thus, the synchronized digital twin models the changing software dynamics, enables adaptive graph learning and supplies a true representation for future reliability analysis using the Temporal Graph Transformer. In Eq. (7) the state of the Digital Twin at time t is represented.

$$DT_t = \{V_t, E_t, F_t\} \quad (7)$$

where$V_t$ denotes the set of software modules, $E_t$ represents the dependency relationships among modules, and $F_t$ contains the runtime feature vectors collected from the executing software.The synchronization process is formulated in Eq. (8)

$$DT_t = \Phi(DT_{t-1}, X_t) \quad (8)$$

where$DT_{t-1}$ is the previous Digital Twin state, $X_t$ represents the newly acquired runtime software information at time t, and Φ(·) denotes the synchronization function that updates the virtual software model.To ensure that the Digital Twin accurately reflects the physical software system, the synchronization error is minimized using the following objective function is presented in Eq. (9)

$$L_{sync} = \frac{1}{N}\sum_{i=1}^{N} \| P_i - DT_i \|_2^2 \quad (9)$$

where $P_i$ denotes the observed state of the physical software module, $DT_i$ represents the corresponding synchronized Digital Twin state, N is the total number of software modules, and $\|.\|_2$ denotes the Euclidean norm. Minimizing $L_{sync}$ continuously aligns the virtual model with the physical software system, enabling the proposed DTAGL-ESRP framework to accurately capture runtime software evolution and improve the robustness of software reliability prediction.Figure 2 shows the digital Twin Synchronization process.

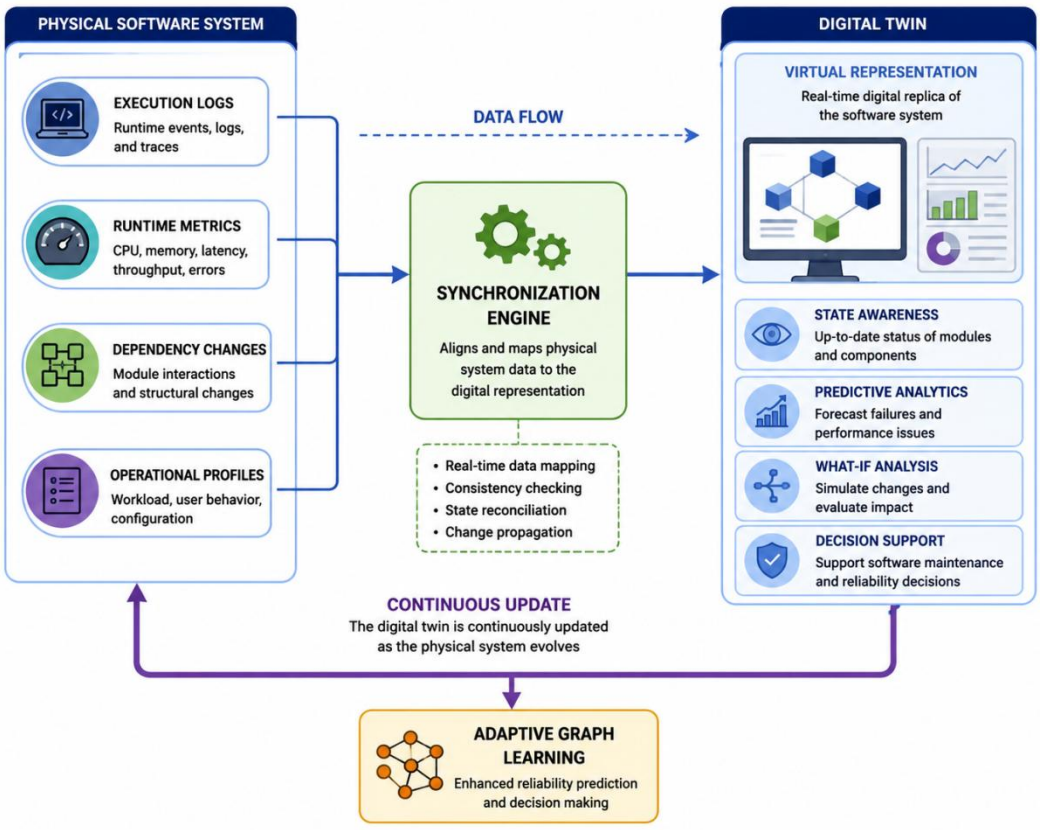



**Figure 2.** Digital Twin Synchronization process

**Dynamic Software Dependency Graph Construction**

The software system is then converted to a dynamic dependency graph, where the software structural relationships are explicitly represented after the synchronization of Digital Twin. The nodes are software components and the arcs are the dependency between nodes, e.g. function calls, inheritance, and package interaction or running time communication. The structure of the graph preserves the structure used in the propagation of defects and failures across the software system as opposed to the conventional vector-based representations. The dependency graph is dynamically rebuilt to capture the latest execution behavior as the Digital Twin is updated, which allows the proposed framework to accurately model the software interactions that change over time in order to reliably predict the software reliability. The software dependency graph can be represented as in Eq. (10)

$$G = (V, E) \quad (10)$$

where$V = \{v_1, v_2, \ldots, v_n\}$denotes the set of software modules and E represents the dependency relationships among them.The adjacency matrix is defined in Eq. (11)

$$A_{ij} = \begin{cases} 1, & \text{if module i depends on module j,} \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

The initial node feature matrix is presented in Eq. (12)

$$H^{(0)} = [x_1, x_2, \dots, x_n]^T \quad (12)$$

where$x_i$ represents the normalized software metric vector associated with module i.

**Temporal Graph Transformer Learning**

The proposed framework uses a Temporal Graph Transformer (TGT) to jointly model structural dependencies and temporal evolution of software. It employs a multi-head self-attention mechanism to learn from the influence of adjacent software modules, and to embed the temporal positional encoding to capture software execution dynamics. This allows the model to detect the patterns of fault propagation, long-range dependencies between modules, and changing behaviour that is difficult to capture with traditional graph neural networks or sequential learning models. As such, more precise graph embedding are created for software reliability prediction. The matrices of Query, Key and Value are calculated in Eq. (13).

$$\mathrm{Q} = \mathrm{HW_Q}, \ \ \mathrm{K} = \mathrm{HW_K}, \ \ \mathrm{V} = \mathrm{HW_V} \quad (13)$$

whereH is the node feature matrix and $\mathrm{W_Q}$, $\mathrm{W_K}$, and $\mathrm{W_V}$ are trainable weight matrices.The attention score between two software modules in Eq. (14)

$$e_{ij} = \frac{Q_i K_j^T}{\sqrt{d}} \quad (14)$$

whered denotes the feature dimension.The normalized attention coefficient is presented in Eq. (15)

$$\alpha_{\mathrm{ij}} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})} \quad (15)$$

The updated graph embedding is obtained in Eq. (16)

$$H' = \sum_j \alpha_{\mathrm{ij}} V_j \quad (16)$$

whereH′ represents the learned node representation used for subsequent reliability prediction.

**Adaptive Concept Drift Handling**

The behavior of software changes continuously, as the code is modified, workload is changing, system is upgraded or environment has changed, causing concept drift which negatively affects the prediction accuracy. In order to maintain good performance, the proposed framework constantly tracks the errors of the predictions and automatically identifies when a behavioural change has occurred. Once a large amount of drift is detected, the model will not be retrained but it will be updated incrementally with the data it has observed at runtime. The mechanism helps the framework to keep its prediction stability during the software's life cycle while minimizing computational burdens. The prediction error at time t is computed in Eq. (17)

$$e_t = \mid y_t - \hat{y}_t \mid \quad (17)$$

where$y_t$ and $\hat{y}_t$ denote the actual and predicted reliability labels, respectively.The moving average prediction error is presented in Eq. (18)

$$\bar{\mathrm{e}}_t = \frac{1}{W} \sum_{\mathrm{i=t-W}}^{t} e_i \quad (18)$$

whereW is the sliding window size.The drift score is calculated in Eq. (19)

$$DS_t = \mid et - \bar{\mathrm{e}}_t \mid \quad (19)$$

Concept drift is detected in Eq. (20)

$$DS_t > \delta \quad (20)$$

whereδ is the predefined drift threshold.Whenever drift is detected, the model parameters are updated in Eq. (21)

$$\theta_{t+1} = \theta_t - \eta \nabla L \quad (21)$$

whereθ represents the model parameters, η is the learning rate, and L denotes the prediction loss.

**Reliability Prediction**

Following adaptive graph learning, the optimized graph embedding are passed on to a fully connected classification layer to measure the degree of trustworthiness for each software module. The classifier is used to classify the reliability of the module based on the integration of the information from structural dependency, temporal execution characteristics, and dynamically updated representations of the software. The end-to-end prediction mechanism allows to precisely and flexibly evaluate the reliability of software in a continuously changing execution context. The predicted reliability probability is computed in Eq. (22)

$$\hat{y} = \sigma(WH' + b) \quad (22)$$

whereW is the trainable weight matrix, b is the bias vector, and σ(·) is the sigmoid activation function defined in Eq. (23)

$$\sigma(z) = \frac{1}{1+e^{-z}}. \quad (23)$$

The prediction model is optimized using the binary cross-entropy loss is presented in Eq. (24)

$$L_{pred} = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)], \quad (24)$$

whereN denotes the total number of software modules, $y_i$ is the ground-truth class label, and $\hat{y}_i$ is the predicted reliability probability. By minimizing $L_{pred}$, the proposed DTAGL-ESRP framework can more accurately identify reliable and defective software modules while still having a high level of generalization performance across a variety of software projects. The hyper parameter values for training the proposed DTAGL-ESRP framework are shown in Table 1. The model was optimized with Adam optimizer, initial learning rate of 0.001, 100 epochs trained with 64 batch size and early stop (patience = 15 epochs) by implementing to prevent over fitting. The Temporal Graph Transformer was designed to take 4 transformer layers, 8 multi-head attention heads, a 256-dimensional hidden layer, a 128-dimensional graph embedding, a 512-dimensional feed-forward network and a dropout rate of 0.30, with ReLU activation functions used for feature learning and Sigmoid activation functions used for final reliability classification.

**Table 1.**Hyper parameter Specifications of the Proposed DTAGL-ESRP Framework

| Hyper parameter | Value |
|---|---|
| Optimizer | Adam |
| Initial Learning Rate | 0.001 |
| Batch Size | 64 |
| Number of Epochs | 100 |
| Early Stopping Patience | 15 |
| Input Feature Dimension | 32 |
| Hidden Dimension | 256 |
| Graph Embedding Dimension | 128 |
| Number of Temporal Graph Transformer Layers | 4 |
| Number of Attention Heads | 8 |
| Feed-Forward Network Dimension | 512 |
| Dropout Rate | 0.30 |
| Activation Function | ReLU |
| Output Activation | Sigmoid |
| Graph Aggregation | Multi-Head Self-Attention |

**Results And Discussion**

This section outlines the experimental assessment of the proposed DTAGL-ESRP framework with the benchmark NASA software defect datasets. The experiments have been performed to evaluate the proposed framework for predicting the software reliability in dynamic execution environment. The performance of DTAGL-ESRP is measured against some state-of-the-art software reliability prediction models using various evaluation metrics such as Accuracy, Precision, Recall, F1-score and Log Loss. Additionally, the adaptability of the framework is assessed by the Digital Twin Synchronization Error, Concept Drift Detection Accuracy, Adaptation Time and Computational Time. The experimental results confirm that the proposed approach has

excellent prediction accuracy, robustness, and interpretability through the integration of the four main components: Digital Twin synchronization, Temporal Graph Transformer learning, adaptive concept drift handling, and explainability using SHAP, and is well suited for real-time software reliability management.

**A. Data set**

In this work NASA (http://mdp.ivv.nasa.gov) dataset is used. There are 22 static metrics of the software that are the data set. The NASA Metrics Data Program (NASA MDP) dataset includes software module data gathered from a number of NASA software development projects written in a variety of programming environments [19-20]. The collection of static software metrics, which represent code complexity, inheritance, coupling, size and design characteristics of each software module, are used to represent each module, with a binary class label indicating the presence or absence of faults in each module. The number of software modules and defect density are different across various projects, including those of CM1, KC1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4 and JM1. These are metrics that can be obtained from the source code and are commonly used in benchmarking algorithms used to predict software defects. The two NASA datasets, MDP and PROMISE, were initially pre-processed using missing value imputation, duplicate deletion, Z-Score normalization, outlier detection and removal, correlation-based feature selection, and class balancing using SMOTE technique. The whole data set was pre-processed and then randomly divided into 80% training and 20% test sets. Five-fold cross validation was used in training to ensure that the proposed model is strong and generalizable, in which four-fold was used to learn the model and one-fold was used to validate the model in each fold. The last test was conducted on the independent 20% test set, thus providing an unbiased assessment of the proposed DTAGL-ESRP framework.

**Performance Metrics**

The effectiveness of the DTAGL-ESRP framework is fully assessed with commonly used classification measures for the software reliability prediction. In particular, the predictive ability, robustness and reliability of the proposed model are evaluated through the use of the following metrics: Accuracy, Precision, Recall, and F1-score. Accuracy refers to the overall correctness of the classification, and Precision and Recall are used to measure the capability of correctly detecting defective software modules and reduce the number of false predictions [21]. While MCC is a strong measure of evaluation as it includes all components of the confusion matrix, the F1 score gives a balanced evaluation of Precision and Recall, especially in imbalanced datasets [22-23]. AUC shows how well the classifier discriminates between classes for various decision thresholds. The presentation of the performance is in the form of Eq. (25) – Eq. (28).

$$Accuracy = \frac{\text{TP+ TN}}{\text{TP+ TN+ FP+ FN}} \quad (25)$$

$$Recall = \frac{\text{TP}}{\text{TP}+FN} \quad (26)$$

$$Precision = \frac{TP}{TP + FP} \quad (27)$$

$$F1Score = \frac{2*(\text{Recall}*\text{Precision})}{\text{Recall+Precision}} \quad (28)$$

The study of the proposed framework, namely DTAGL-ESRP is carried out in this paper with the help of Table 2, where each major component is examined, and its effect is analysed.The most severe performance drop occurs when the Digital Twin module is removed, as this leaves the system with less ability to maintain continuous software synchronization, whereas excluding the Temporal Graph Transformer or Concept Drift Handling also leads to a drop in prediction effectiveness, since it reduces dependency modelling and adaptive learning capabilities.The overall performance of the complete DTAGL-ESRP framework is best with an accuracy of 98.56%, recall of 98.75%, and precision of 98.26% and F-measure of 98.50% which shows that each architectural component adds to the improvement of software reliability prediction.

**Table 2.** Ablation Study of the proposed method

| Model Configuration | Accuracy (%) | Recall (%) | Precision (%) | F-Measure (%) |
|---|---|---|---|---|
| Without Digital Twin | 95.84 | 95.61 | 95.72 | 95.66 |
| Without Temporal Graph Transformer | 96.41 | 96.52 | 96.34 | 96.43 |
| Without Concept Drift Handling | 97.18 | 97.06 | 97.12 | 97.09 |
| Without SHAP Explainability | 97.82 | 97.64 | 97.71 | 97.67 |
| Proposed DTAGL-ESRP | 98.56 | 98.75 | 98.26 | 98.50 |

Table 3 shows the comparative performance of the proposed DTAGL-ESRP framework with the other software reliability and defect prediction methods. The proposed model has the highest accuracy of 98.56%, recall of 98.75%, precision of 98.26% and F-measure of 98.50% which is better than Decision Tree, VESDP, WFS, and Bi-LSTM models. The improvements have shown that the fusion of the dynamic software dependency and runtime behavior captured by the Digital Twin synchronization, with Temporal Graph Transformer-based adaptive graph learning, is an effective method for achieving the goal of more accurate and reliable software reliability prediction. Figure 3.shows the radar chart of the proposed framework with existing method.

**Table 3**. Comparative Performance of the proposed method

| **Methods** | **Accuracy** | **Recall** | **Precision** | **F-Measure** |
|---|---|---|---|---|
| Decision Tree[6] | 97.22 | 96.58 | 97.04 | 96.98 |
| VESDP [7] | 79.92 | 94.4 | 92.01 | 85.63 |
| WFS [9] | 92.36 | 93.21 | 92.85 | 91.25 |
| Bi-LSTM [10] | 88.00 | 48.00 | 57.00 | 51.00 |
| This work | 98.56 | 98.75 | 98.26 | 98.50 |

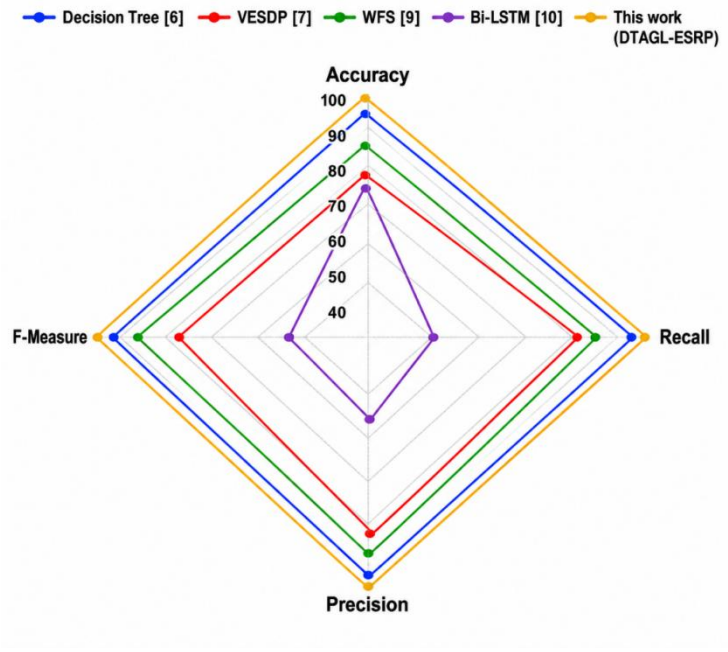



**Figure 3**. Radar chart of the proposed framework with existing method

Table 4 shows the performance of the proposed DTAGL-ESRP framework's Digital Twin synchronization module.The Digital Twin is highly synchronized with the software system with a synchronization accuracy of 99.12% and a low synchronization error of 0.008.Additionally, the framework's performance shows an average synchronization time of 17.4 ms and a runtime update frequency of 12.8 Hz, which proves its ability to update software states in real-time for real-time reliability prediction.

**Table 4.** Digital Twin Performance Evaluation

| Metric | Value |
|---|---|
| Digital Twin Synchronization Accuracy (%) | 99.12 |
| Synchronization Error | 0.008 |
| Average Synchronization Time (ms) | 17.4 |
| Runtime Update Frequency (Hz) | 12.8 |

Table 5 shows the results of the SHAP-based explainability analysis for the proposed DTAGL-ESRP framework, represented in the table by the value of the software metrics. The results of the SHAP-based explainability analysis for the proposed DTAGL-ESRP framework, represented in the table as the value of software metrics, are presented.The software complexity and module coupling have the greatest influence on the reliability assessment, as shown by the Cyclomatic Complexity (0.246) and Lines of Code (0.213).SHAP

analysis provides greater transparency of the model, highlighting the most important software metrics and helping to make informed decisions on software maintenance and debugging.

**Table 5.**Explainability Analysis (SHAP)

| Software Metric | Mean SHAP Importance |
|---|---|
| Cyclomatic Complexity | 0.246 |
| Lines of Code (LOC) | 0.213 |
| Coupling Between Objects (CBO) | 0.191 |
| Response for Class (RFC) | 0.158 |
| Lack of Cohesion (LCOM) | 0.117 |
| Weighted Methods per Class (WMC) | 0.075 |

The computational performance results of the proposed DTAGL-ESRP framework are reported in Table 6.The model can be trained in 26.8 minutes and tested in 5.4 seconds. An average inference time of 2.1ms per sample shows the model's suitability for real-time software reliability prediction.Moreover, it takes advantage of 6.7 GB of GPU memory and 3.42 million trainable parameters, thus achieving a balance between computational efficiency and prediction accuracy.

**Table 6.** Computational Performance

| Metric | Value |
|---|---|
| Training Time (min) | 26.8 |
| Testing Time (s) | 5.4 |
| Average Inference Time (ms/sample) | 2.1 |
| GPU Memory Utilization (GB) | 6.7 |
| Total Model Parameters | 3.42 Million |

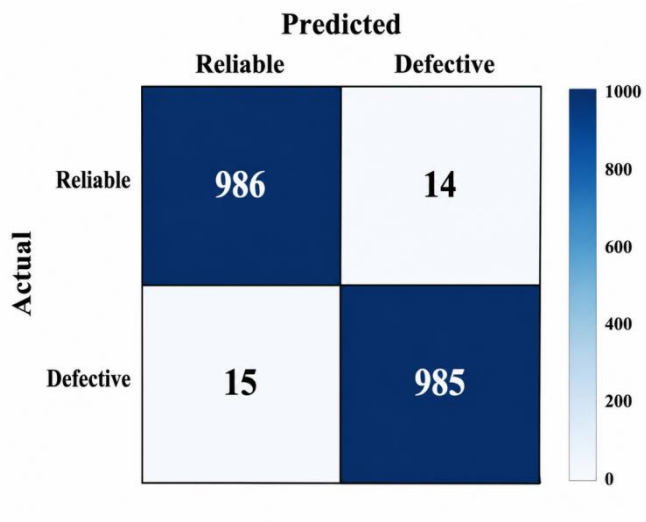



**Figure 4.** Confusion of the proposed method

The confusion matrix for the proposed TAGL-ESRP framework is shown in Figure 4.The model was able to correctly classify the software module as reliable with 986 cases and as defective with 985 cases, while also misclassifying 14 software modules as reliable (false positives) and 15 as defective (false negatives).Based on these results, the proposed framework can be seen to achieve a good balance of prediction performance and minimum misclassification errors to separate reliable and faulty software modules.

**Conclusion**

The paper proposes a DTAGL-ESRP framework to cope with the drawbacks of traditional software reliability prediction techniques in dynamic software systems.The proposed framework unifies continuous Digital Twin synchronization, dynamic construction of software dependency graph, a Temporal Graph Transformer for adaptive graph representation learning, concept drift detection and SHAP-based explainability into a comprehensive end-to-end framework.The framework can effectively model changing inter-module dependencies, and can continuously synchronize the behavior of the software at runtime with its virtual

counterpart, effectively capturing software evolution and enabling software engineers to receive transparent reliability assessments of the software.The benchmark NASA dataset was used to test the proposed DTAGL-ESRP framework.The experimental results showed that this model achieved the highest classification accuracy of 98.56%, recall of 98.75%, precision of 98.26%, and F-measure of 98.50%, which are superior to Decision Tree, VESDP, WFS and Bi-LSTM based approaches.The ablation study also highlighted the contributions of each component of the architecture and demonstrated that the combination of these components, namely Digital Twin synchronization, Temporal Graph Transformer learning, adaptive concept drift handling, and explainability using SHAP led to the best prediction performance.Furthermore, Digital Twin module had a synchronisation accuracy of 99.12% and a synchronization error of 0.008 with the data from the real system, and the SHAP analysis results showed that the most influential software metrics on reliability prediction are Cyclomatic Complexity, Lines of Code and Coupling between Objects.Furthermore, the computational assessment showed that the framework is also efficient for practical use, with only 26.8 minutes needed for model training and an average of 2.1 milliseconds per sample for inference.Future research efforts will be centred on scaling the proposed framework to large-scale distributed software systems, cloud-native applications and micro service architectures as well.Future work will focus on federated Digital Twin learning, graph foundation models and continual learning strategies for improving scalability, robustness and generalisation within diverse software development environments.

**References**


1. Samal, Umashankar, and Ajay Kumar. "A neural network approach for software reliability prediction." *International Journal of Reliability, Quality and Safety Engineering* 31, no. 03 (2024): 2450009.

2. Samal, Umashankar, and Ajay Kumar. "Enhancing software reliability forecasting through a hybrid ARIMA-ANN model." *Arabian Journal for Science and Engineering* 49, no. 5 (2024): 7571-7584.

3. Alyahyan, Saleh, Mohammed NaifAlatawi, Mrim M. Alnfiai, ShoayeeDlaimAlotaibi, Abdullah Alshammari, ZaidAlzaid, and HathalSalamahAlwageed. "Software reliability assessment: An architectural and component impact analysis." *Tsinghua Science and Technology* 30, no. 2 (2024): 908-925.

4. Gandikota, Hari Prasad, Abirami S, and S Kumar M. "CT scan pancreatic cancer segmentation and classification using deep learning and the tunicate swarm algorithm." Plos one 18.11 (2023): e0292785.

5. Yang, Minghao, Shunkun Yang, and Chong Bian. "Software Reliability Prediction by Adaptive Gated Recurrent Unit-Based Encoder-Decoder Model With Ensemble Empirical Mode Decomposition." *Software Testing, Verification and Reliability* 34, no. 8 (2024): e1895.

6. Habtemariam, GetachewMekuria, Sudhir Kumar Mohapatra, HussienWorkuSeid, Srinivas Prasad, Tarini Prasad Panigrahy, and Prasanta Kumar Bal. "Prediction and classification of software reliability using ensemble learning." *Journal of Integrated Science and Technology* 13, no. 2 (2025): 1026-1026.

7. Ali, Misbah, TehseenMazhar, YasirArif, Shaha Al-Otaibi, YazeedYasinGhadi, Tariq Shahzad, Muhammad Amir Khan, and HabibHamam. "Software defect prediction using an intelligent ensemble-based model." *IEEe Access* 12 (2024): 20376-20395.

8. Abdu, Ahmed, ZhengjunZhai, Hakim A. Abdo, RedhwanAlgabri, Mohammed A. Al-Masni, MannanSaeed Muhammad, and YeongHyeonGu. "Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model." *Scientific Reports* 14, no. 1 (2024): 14771.

9. Kumar, M. S, and D. Harshitha. "Process innovation methods on business process reengineering." Int. J. Innov. Technol. Explor. Eng 8.11 (2019): 2766-2768.

10. Natarajan, V. Anantha, and M. S Kumar. "Improving qos in wireless sensor network routing using machine learning techniques." 2023 International Conference on Networking and Communications (ICNWC). IEEE, 2023.

11. Khalid, Aimen, Gran Badshah, NasirAyub, Muhammad Shiraz, and Mohamed Ghouse. "Software defect prediction analysis using machine learning techniques." *Sustainability* 15, no. 6 (2023): 5517.

12. Balaji, K., P. SaiKiran, and M. S Kumar. "Power aware virtual machine placement in IaaS cloud using discrete firefly algorithm." Applied Nanoscience 13.3 (2023): 2003-2011.

13. Siddiqui, Tamanna, and MohdMustaqeem. "Performance evaluation of software defect prediction with NASA dataset using machine learning techniques." *International Journal of Information Technology* 15, no. 8 (2023): 4131-4139.

14. Kumar, M. Sunil, et al. "Automated Extraction of Non-Functional Requirements From Text Files: A Supervised Learning Approach." Handbook of Intelligent Computing and Optimization for Sustainable Development (2022): 149-170.

15. Sasikumar, S., and C. Dhilipan. "Predicting Software Reliability and Maintainability Using Ensemble Learning Techniques and Source Code Metrics." In *Doctoral Symposium on Computational Intelligence*, pp. 771-787. Singapore: Springer Nature Singapore, 2025.

16. AnanthaNatarajan, V., M. S Kumar, and V. Tamizhazhagan. "Forecasting of wind power using LSTM recurrent neural network." Journal of Green Engineering 10 (2020).

17. Kumar, N. Anand, and E. K. Girisan. "A Deep Learning Framework for Software Reliability Using AST Correlations." In *2025 9th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pp. 1182-1188. IEEE, 2025.

18. Samal, Umashankar, and Ajay Kumar. "A software reliability model incorporating fault removal efficiency and it's release policy." *Computational Statistics* 39, no. 6 (2024): 3137-3155.

19. Pradhan, Vishal, ArijitPatra, Ankush Jain, Garima Jain, Ajay Kumar, JoydipDhar, AnjanBandyopadhyay, SauravMallik, Naim Ahmad, and Ahmed Said Badawy. "PERMMA: Enhancing parameter estimation of software reliability growth models: A comparative analysis of metaheuristic optimization algorithms." *Plos one* 19, no. 9 (2024): e0304055.

20. Kumar, M. S, and K. JyothiPrakash. "Internet of things: IETF protocols, algorithms and applications." Int. J. Innov. Technol. Explor. Eng 8.11 (2019): 2853-2857.

21. Hussain, Adel Sufyan, ZedanZubairMashikhin, and NooraTahaAbduirazzaq. "Modified Artificial Neural Networks in Stochastic Differential Equations to Estimate the Occurrence Rate of a New Process for Software Reliability Growth Modelling." *Baghdad Science Journal* 22, no. 11 (2025): 3825-3833.

22. Sangamithra, B., P. Neelima, and M. S Kumar. "A memetic algorithm for multi objective vehicle routing problem with time windows." 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE). IEEE, 2017.

23. Vashishtha, Aditi, and Shiv Kumar Sharma. "Estimation of Software Reliability Testing Using Machine Learning Techniques." In *2025 International Conference on Intelligent Control, Computing and Communications (IC3)*, pp. 502-506. IEEE, 2025.