Policy Research Journal
ISSN (E): 3006-7030 ISSN (P) : 3006-7022 Volume 4, Issue 3, 2026

PREDICTIVE ANALYTICS FOR PROJECT RISK MANAGEMENT USING MACHINE LEARNING

Mahnoor Zaman[1], Nasir Umar[*2], Amna Afzal[3], Afifa Saif[4]

[1,2,3,4,] *NFC Institute of Engineering and Technology, Multan, Pakistan*

[1]786mahnoorzaman@gmail.com, [*2]nasir.umar@nfciet.edu.pk, [3]amnaafzal605@gmail.com
[4]saifafifa52@gail.com

DOI: https://doi.org/10.5281/zenodo.19384197

Keywords

Article History
Received: 31 January 2026
Accepted: 15 March 2026
Published: 31 March 2026

Corresponding Author: *
Nasir Umar

Abstract

Software projects frequently encounter challenges, including schedule delays, cost overruns, quality defects, scope creep, resource shortages, and reduced team productivity. Accurate estimation of project duration and cost, based on project length and complexity, is essential for successful and timely completion. Previously, managers assessed risks and project duration based on their own experience. While greater experience often led to better risk management, this approach was often slow, inaccurate, and subject to bias. The solution is to combine predictive analysis and machine learning. Rather than relying on manual estimation, leveraging machine learning offers fast, accurate, and unbiased prediction. To achieve this, the model is trained on historical data collected from different organizations using machine learning. This data may include project size, actual cost, team details, resource data, requirements, technical complexity factors, quality and testing factors, project management metrics, risk-related variables, and outcome labels. Together, these factors form a powerful dataset for model training. The model generates a prediction based on given inputs. Although machine learning has been widely used in software projects to predict development aspects such as time, cost, and potential risks, few approaches have integrated these elements within a single framework. This study introduces an optimized Deep Neural Network model that unifies time, effort, and cost estimation with risk prediction and offers risk mitigation strategies to enhance project outcomes. The model initially predicts risks and subsequently recommends specific actions to prevent them. Rather than estimating cost and time in isolation, the model incorporates risk predictions to refine time, cost, and effort estimates, providing targeted recommendations for risk mitigation. The proposed model offers a practical, AI-driven approach for predicting risks and identifying effective mitigation strategies. It delivers reliable and accurate risk-aware forecasts, as well as preventive measures, to enhance project outcomes, strengthen project planning, and reduce failure rates.

INTRODUCTION

Efficiency and proper management of software projects are the major challenges in the IT sector. It is really important to complete the project on time and under the budget. The IT sector has made good progress, but it still needs to improve the efficiency of software projects and proper management. This sector is using different emerging technologies, but is far behind in the successful development of software projects. The project fails, but still costs millions of dollars due to improper

management. Proper management and timely risk identification may prevent these obstacles. The IT sector is improving day by day, but there is still a significant need for improvement in software project management. There is a need for two terminologies for the successful completion of projects: risk identification and management. If risk identification is done on time, then these obstacles can be prevented, and the project can be completed successfully before the deadline.

Risks usually arise from changes in requirements, poor communication, inaccurate estimation, and other factors. It all depended on the management. A few years ago, project estimates relied on the manager's experience and judgment. Experts used heuristics, rule-based models, and basic statistics for decision-making. If the manager has good experience, his estimate could be right, but not certain. This estimation could be inaccurate, biased, and slow. A minor malfunction may lead to the project's failure. [1] AI has become a trusted companion in the path towards risk mitigation. As AI plays a crucial role in other fields, it can also help with management and risk identification. AI models can predict anomalies and risks even before the project commences. AI can determine more patterns and deviations that may be missed by humans, enabling the organization to make accurate decisions. The solution to resolving risk identification and accurate estimation is using machine learning models to predict. Machine learning models can analyze a large amount of data at a time and can detect more anomalies than humans.

Machine learning has enhanced software risk assessment by increasing prediction accuracy through training on historical data and the automatic identification of patterns, structures, and trends without human intervention. Various machine learning models have been utilized in multiple studies. Initial approaches employed supervised learning models, including decision trees, support vector machines, and the Naive Bayes classifier. These models were trained on historical logs, past bug reports, and project metrics; however, they did not achieve satisfactory accuracy. While training accuracy was high, the models failed to demonstrate strong performance when applied to new projects [9].

To address previous limitations, the Gradient Boosting Machine and Support Vector Machine models were introduced. These models effectively handle imbalanced data, capture non-linear relationships, and demonstrate greater accuracy and stability compared to earlier approaches. As a result, predictive accuracy improved significantly. As project data evolved, new modeling challenges arose. In response, neural network models were employed to analyze time-series data. Specifically, researchers investigated Recurrent Neural Network and Long Short-Term Memory models, which can detect failure risks in real time and capture temporal dependencies. However, these models lacked transparency, making it difficult for developers and managers to understand the rationale behind risk predictions. This lack of interpretability hindered industry adoption [9].

Ensemble learning techniques are subsequently employed to combine multiple classifiers. Hybrid ensemble models reduce errors by leveraging the strengths of diverse algorithms while minimizing the limitations of individual methods. Further improvements may be achieved by integrating additional data sources, including code metrics, logs, bug reports, team activity, time, and cost. The application of advanced feature engineering and model interpretability techniques can also enhance model performance. To overcome the limitations of existing models, this study presents a deep neural network (DNN)-based software risk mitigation model that learns complex and non-linear relationships among project risk factors, which are often overlooked by traditional statistical models and shallow machine learning techniques. The model is trained on historical project data utilizing deep representation, regularization, and interpretability methods, thereby enhancing accuracy and adaptability in software project management contexts.

The key purpose of ML models is to provide an accurate appraisal and analysis, in contrast to traditional methods.

[2] The following steps are used to perform the predictive analysis:

1. Define the Problem

The primary objective should be clearly defined, specifying the desired outcomes and providing a

detailed discussion of the dataset used for model training.

2. Gather historical data

The dataset should be compiled from historical and original data sourced from various organizations, potentially including fields such as time duration, budget, requirements changes, team size and experience, bugs, risks, failures, and technology used.

3. Data cleaning and preprocessing

As the dataset forms the foundation for prediction, it is essential to address uncertainties and missing values by cleaning the data, removing incomplete entries, correcting inconsistencies, encoding categorical variables, and eliminating outliers where necessary.

4. Feature Engineering

To improve model efficiency, it is important to select the most relevant attributes and engineer new, meaningful features, such as complexity scores and risk levels. Feature engineering should be performed to enhance model performance.

5. Split the Dataset

The cleaned dataset should then be divided into training and testing subsets, with the ratio determined by the developer or based on the structure of pre-existing datasets. In some cases, a separate validation set may be used.

6. Choose a Predictive Model

The selection of an appropriate predictive model depends on the project's nature and the required outcomes. Various machine learning algorithms, including Logistic Regression, Support Vector Machine (SVM), Random Forest, and Gradient Boosting, can be employed to capture complex features in historical data and provide consistent predictive performance. For more complex datasets, deep learning approaches such as neural networks may be utilized, as they can extract intricate patterns and learn non-linear relationships.

7. Train the Model

The model is trained on the prepared data, learning patterns from historical records, and hyperparameters are adjusted to optimize accuracy.

8. Test & Validate the Model

Evaluate the trained model on test or validation data to assess its ability to generalize to new data. Measure performance using metrics such as accuracy, precision, recall, and F1 score. These metrics indicate prediction accuracy and help minimize false results.

Literature Review

Software project failures result in a loss of $122 million for every $1 billion invested by enterprises. Organizations often do not receive value proportional to the cost and effort expended. Approximately 44% of projects fail due to factors such as inadequate planning and estimation, insufficient requirement analysis, and ineffective risk management. Risks frequently emerge from issues, including the addition of new features without adequate planning or control over time and cost. Consequently, the final project often fails to meet user expectations [11]. Artificial intelligence (AI) has significantly influenced numerous fields, with software engineering being among the most impacted domains. AI has enhanced the Software Development Life Cycle (SDLC) by automating decision-making, improving predictive capabilities, reducing human error, and facilitating data-driven planning throughout all development phases. Software engineering projects frequently encounter challenges such as schedule overruns, cost overruns, and evolving requirements [3]. Early risk anticipation allows teams to identify potential issues, including cost overruns, schedule delays, uneven resource allocation, and inadequate planning. These risks can be mitigated through improved planning and more effective resource distribution, thereby reducing the likelihood of software failure. Consequently, risk anticipation has become an essential component of the SDLC rather than merely a precautionary measure.

Previous studies have employed machine learning, deep learning, and various traditional models for detecting software project risks. Prior research has demonstrated that these models can achieve high predictive accuracy [3]. Artificial Neural Networks (ANN) and Random Forest algorithms have outperformed other approaches in managing complex risks. Support Vector Machines (SVM) have achieved an accuracy of 80%, with 80% recall, 84% precision, and an 80% F1-score [4]. This model utilized t-distributed Stochastic Neighbor Embedding (t-SNE) for feature engineering, rather than traditional Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA),

to better preserve non-linear relationships in project data. Various models have been operated on different attributes. AI-driven models have surpassed traditional models.

Although previous studies have reported promising results, several limitations persist. Many approaches prioritize high predictive accuracy while neglecting the integration of additional relevant factors. Furthermore, some studies utilize domain-specific and small datasets, which restrict generalizability [4, 8, 10]. The majority of research emphasizes classification performance, offering limited analysis of cost and schedule considerations [6, 8]. Risk prediction is frequently examined in isolation, without integration with cost and time, thereby reducing its practical value for project planning. Additionally, some studies address single-task prediction, such as individual risk components, but fail to incorporate these risks into a comprehensive risk management framework [5, 7]. The table given below presents a detailed review of previous studies:

Table 1: Limitations of the literature

Reference	Dataset/Techniq ue	Contribution/Me thod	Result	Limitations
[4]	Historical project data (schedule, task, resources, outcomes), ML	Dimensionality reduction using t-SNE, Gradient Boosting Machine	Resource utilization improved to 85%, Project cost reduction of 10%	The dataset mainly focuses on time, cost, and resource factors.
Tran et al. (2023) [5]	Multiple project metrics, AI models	Systematic study of ML/DL for software effort estimation	AI models outperform traditional methods in effort estimation	Lacks integration with risk prediction and mitigation strategies
Mahmud et al. (2022) [6]	16 SRP articles (2007–2022)	SLR on software risk prediction using ML	ML-based risk models are effective in early risk identification	Limited recent studies; focus on classification performance only
Bhatia (2024/2025) [7]	Random Forest, LSTM	Application of ML/DL for effort estimation	ML/DL models detect complex patterns and improve prediction metrics	Does not address risk mitigation or dynamic thresholds
Bauskar et al. (2024/2025) [8]	Historical project data + GBM + feature engineering	Predictive analytics with t-SNE/GBM for project risk forecasting	Higher recall and accuracy than conventional approaches	Focuses mostly on classification; limited cost/schedule estimation insights

Polu (2024) [9]	ML, SHAP/LIME explainability	ML-based risk prediction framework with interpretability	Early failure detection with explainability using SHAP/LIME	Limited scalability Diversity unclear
Shabbir et al, 2026 (jCIM) [10]	Public dataset (299 instances, 13 features); ML classifiers (NB, KNN, LR, SVM, MLP); Feature Selection (ANOVA, Chi^2, MI, RFE); SMOTE	Proposed an empirical ML-based framework for early software risk prediction; applied extensive preprocessing, feature selection, and comparative evaluation of classifiers	SVM with mutual information achieved the best performance (~80% accuracy, 84% precision, 80% recall, ~81% F1-score); feature selection significantly improved prediction accuracy	Experiments conducted on a single dataset; default hyperparameters used without tuning, no deep learning or cross-project validation, which may limit generalizability
Roy [12]	Identifying and analyzing risks in construction projects	Risk matrix, analysis of unstructured text and image data.	examines the role of machine learning in advancing digitalization within the construction sector, with a particular focus on the need for specialized expertise and the challenges of data security.	Limited generalizability due to project-specific datasets.
Unnamed Authors [13]	Predicting software project failure.	Logistic Regression (LR), Naive Bayes (NB), Support Vector Machine (SVM), Decision Trees (DT), Neural Networks, Adaptive Neuro	Introduces a robust risk assessment model suitable for application to software projects at any stage of the software development lifecycle.	The model may omit the real-world complexities and certain project variables.

The comparative analysis presented in the table indicates that existing studies offer valuable insights; however, their applicability is often restricted to specific domains and phases. Several studies utilize static and limited datasets, which are insufficient for addressing the dynamic nature of software development. The predominant limitation across most research is the reliance on small, domain-specific, and imbalanced datasets, which contributes to challenges related to generalizability and scalability.

Methodology

This study employs the AEEEM dataset, a resource commonly used for software defect prediction and project risk estimation. The dataset comprises 62 attributes per software module, including 61 software metrics (e.g., CK metrics, object-oriented metrics, code metrics, and change metrics), as well as a single target variable (bug). The AEEEM dataset facilitates module-level prediction, effort and cost estimation using normalized software metrics, and schedule risk analysis through probability-based categorization. The dataset is partitioned into training and testing subsets in a 70:30 ratio. Stratified sampling is used to preserve the original class distribution. Data profiling techniques, including statistical, correlation, and missing value analyses, are applied to ensure data quality and prepare the dataset for training. Following profiling, all features are scaled to ensure equal contribution during model training. Differences in scale between metrics such as Lines of Code (LOC) and Lack of Cohesion of Methods (LCOM) may introduce bias into the model. To address this issue, standardization (z-score normalization) is applied. This method subtracts the mean to center the data and divides by the standard deviation to scale it. As a result, each feature attains zero mean and unit variance [14].

$$X_{Scaled} = \frac{\{X - \mu\}}{\sigma}$$

Software defect datasets, such as AEEEM, are often class-imbalanced, with substantially fewer defective modules (bug=1) than non-defective ones (bug=0). Training a model on imbalanced data without correction can result in predictions biased toward the majority (non-defective) class. To address this, class weighting is implemented during model training. This technique assigns lower importance to the majority classes and greater importance to the minority classes, thereby increasing attention to underrepresented groups. By implementing this approach, class weights can be automatically adjusted according to sample distribution, which may enhance model performance for minority classes [15].

$$w_j = \frac{n_{samples}}{n_{classes} \cdot n_j}$$

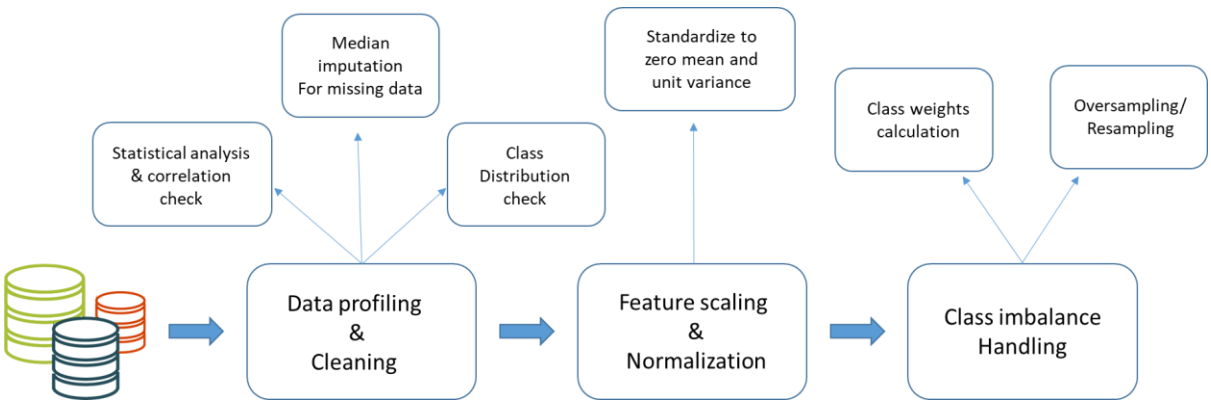

Figure 1: Methodology diagram

A sequential deep neural network is designed with several fully connected layers, each decreasing in size. The ReLU activation function is applied to the hidden layers to introduce non-linearity, while batch normalization is incorporated to accelerate training. Dropout layers are added after each hidden layer to mitigate overfitting. The output layer contains a single neuron with a sigmoid activation function, which predicts the probability that a module is buggy. The network is trained using the Adam optimizer and the binary cross-entropy loss function. A batch size of 32 is utilized, with training conducted for a maximum of 150 epochs. Early stopping with a patience of 10 is implemented to terminate training when performance improvement ceases. A learning rate scheduler is employed to facilitate model convergence and enhance generalization. Additionally, 20% of the data is reserved for evaluating generalization during training.

The trained model is evaluated on unseen data using performance metrics such as accuracy, F1-score, precision, recall, and the area under the ROC curve (AUC). These metrics offer a comprehensive assessment of both overall and minority class prediction performance. A confusion matrix is utilized to visualize true positives, false positives, true negatives, and false negatives, thereby providing detailed insight into the model's effectiveness in detecting buggy modules.

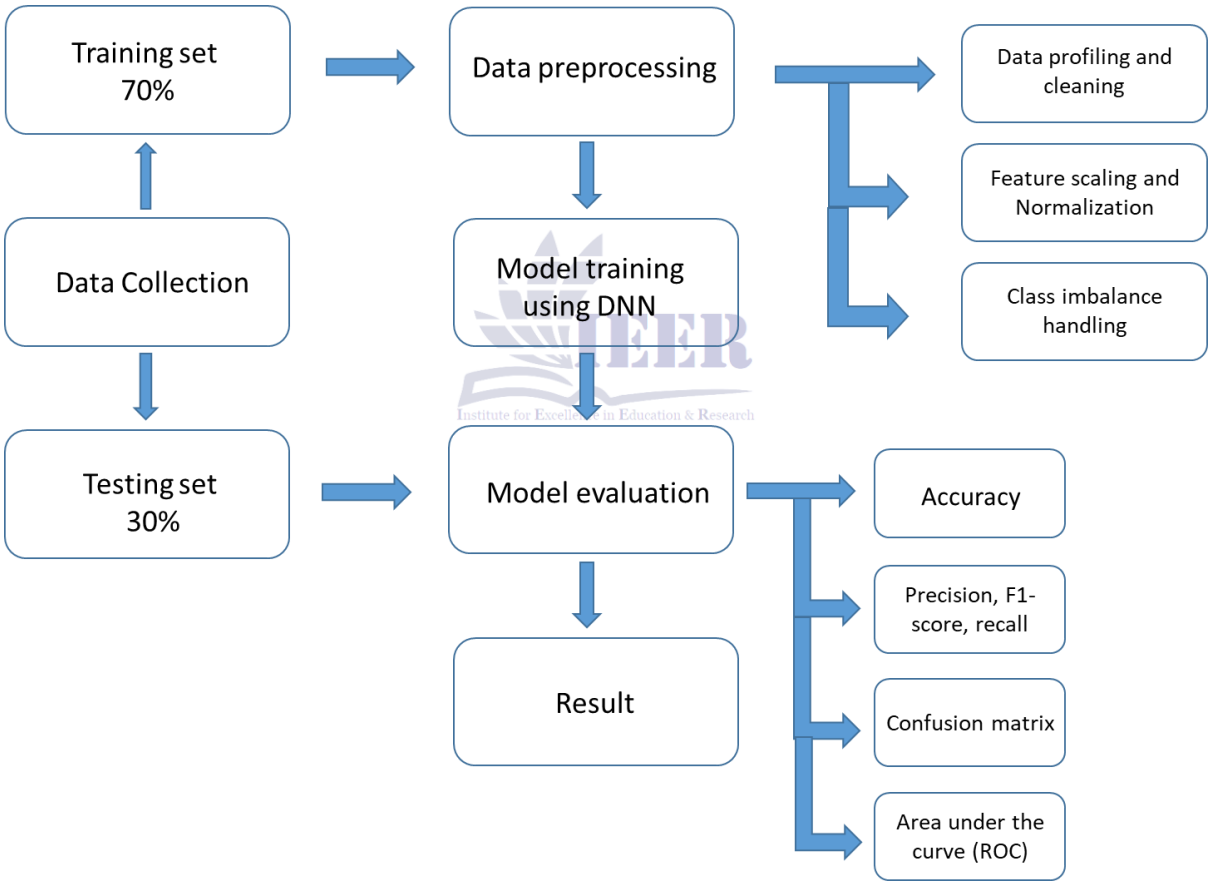

Figure 2: Work Flow Diagram

Results and discussion

The experiments were implemented in Python using libraries such as NumPy, pandas, scikit-learn, imbalance-learn, Matplotlib, Seaborn, and SciPy. The research was conducted on a local laptop equipped with an Intel Core i7-9850H CPU (2.60 GHz), 16 GB of RAM, a 512 GB SSD, and an NVIDIA GeForce MX150 GPU with 2 GB of dedicated memory, providing sufficient computational resources for efficient model training and evaluation. After preprocessing, the **Aeeem** dataset still contains the same number of features as the original dataset, excluding the defect label, which was

subsequently standardized and used for model training and evaluation.

Table 1 provides a comparison of the performance of four machine-learning models, measured without using any feature selection methods. The Deep Neural Network (DNN) was selected as the best among the assessed classifiers due to its high classification accuracy of 91.87% and the highest recall of 99.31%, indicating that it has the greatest ability to detect defective software modules. Conversely, Support Vector Machine (SVM) showed the lowest accuracy of 81.2, whereas Random Forest and XGBoost demonstrated competitive performance, albeit lower than the proposed model.

XGBoost and SVM had slightly high values in terms of precision, but the proposed DNN had a more balanced trade-off between the precision and recall, leading to the best F1-score of 95.62%. Additionally, the DNN achieved a good AUC of 92.52%, demonstrating its effectiveness in classification. These findings demonstrate the usefulness of the proposed strategy in estimating software projects and reducing the threat at risk in the case of training on all possible features.

Table 2: A comparison of the performance of four machine-learning models

Model	Accuracy	Precision	Recall	F1-score	AUC
DNN (Proposed)	0.9187	0.9220	0.9931	0.9562	0.9252
Random Forest	0.859	0.987	0.853	0.915	0.951
SVM	0.812	0.990	0.798	0.884	0.918
XGBoost	0.854	0.993	0.843	0.912	0.957

Figure 3 presents the confusion matrix obtained from the test data of the Deep Neural Network (DNN)-based framework designed to predict software project risks and estimate efforts. The matrix captures classification performance for two risk categories: Risk-Prone (Buggy/High-Risk) and Low-Risk (Non-Buggy) project modules. The results indicate a high false-alarm rate, with the model correctly identifying 156 low-risk cases and falsely classifying 15 low-risk cases as risky. For the high-risk category, the model correctly identified 1171 risky software modules but misclassified 270 risky modules as low risk. Overall, the proposed model achieved 1327 correct predictions and 285 misclassifications. The results of this study validate the efficacy of the streamlined deep neural network (DNN) in identifying high-risk projects, which require proactive mitigation and sound project planning. The low false positive rate ensures that stable projects are not subjected to unnecessary mitigation measures, while the false negative rate indicates that mitigation could be improved through additional hyperparameter tuning of threshold values or enhanced feature representations.

Compared to classical machine learning methods such as Random Forest, Support Vector Machine, and XGBoost, the DNN demonstrated superior discrimination capability, as evidenced by higher accuracy, F1-score, and AUC values reported earlier. Furthermore, the model-generated risk probabilities were effectively integrated into the effort, cost, and schedule estimation pipeline, enabling projects to be classified by schedule risk into high, medium, and low categories, thereby facilitating data-driven mitigation recommendations. In general, the analysis of the confusion matrix supports the idea that the suggested unified DNN framework is a stable and useful tool in estimating and managing software projects based on risk.

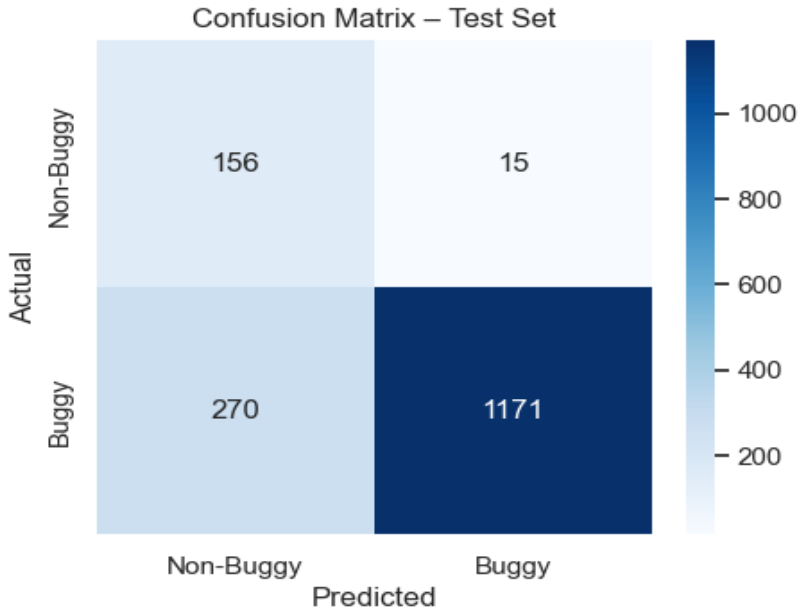

Figure 3: confusion Matrix of Our proposed DNN model

Figure 4 presents the training and validation accuracy and loss curves for the proposed Deep Neural Network (DNN)-based framework, which predicts software project risks and estimates effort. The accuracy curves increase rapidly during the initial training stages and stabilize at approximately 92%. Both loss curves decrease consistently and converge to similar values, indicating stable optimization. The final performance metrics confirm the absence of overfitting. The model achieved training and validation accuracies of 92.98% and 92.02%, respectively, with corresponding training and validation losses of 0.1670 and 0.1774. The minimal difference between training and validation performance demonstrates strong generalization capability. Additionally, the test accuracy of 91.75% is consistent with the validation results, providing further evidence of the model's effectiveness on unseen data.

The findings demonstrate that the optimized deep neural network (DNN) architecture, combined with class weighting, resampling, and dynamic thresholding methods, effectively learns discriminative patterns in software project measures without overfitting to the training set. Consequently, the model is well-suited for deployment in real-world software engineering contexts, particularly for risk-aware effort, cost, and schedule estimation as well as proactive mitigation planning.

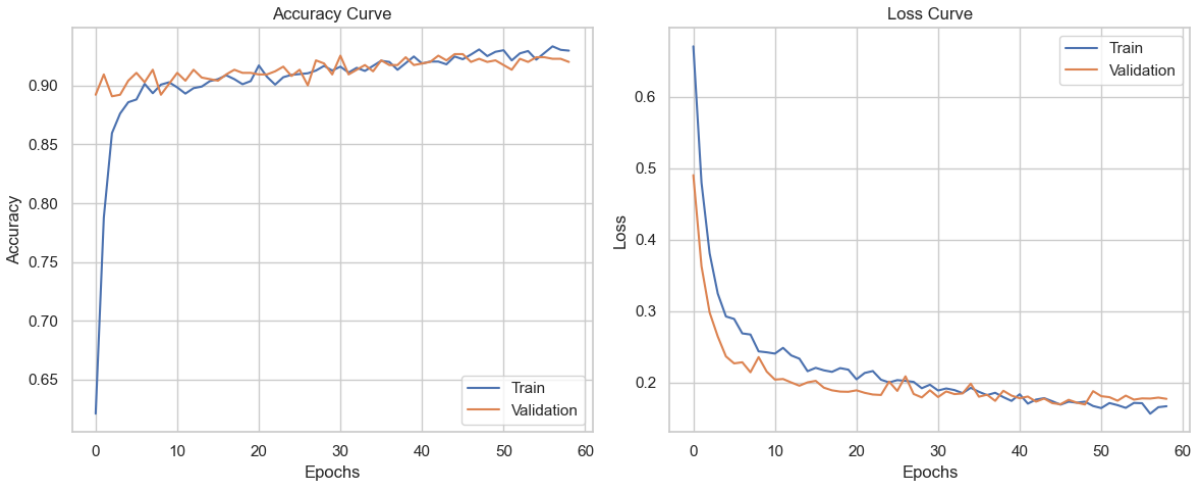

Figure 4: Model Accuracy and Model Loss Curves of DNN Model

Conclusion

This study introduces an AI-driven software project risk management model designed to estimate risk, time, effort, and cost. The model employs a deep neural network to capture the complex and non-linear relationships among software metrics in the AEEEM dataset. Predicted risk probabilities are integrated with cost and effort metrics to support categorizing schedule risk. The model also provides risk-mitigation recommendations based on the assessed risk level. The proposed model offers comprehensive information regarding schedule, cost, and effort estimation, and classifies risk into low, medium, and high levels. In addition to these estimations, the model includes recommendations for mitigating risks at each level. While this model offers a comprehensive solution, several enhancements are possible. The dataset may be expanded, and the model could be applied to larger and more diverse software projects. Additionally, developing a live dashboard would enable continuous updates of risk, effort, and cost estimations throughout the software development process.

REFERENCES

[1] https://community.trustcloud.ai/docs/grc-launchpad/grc-101/risk-management/risk-mitigation-strategies-the-role-of-artificial-intelligence-in-enhancements/#Overview

[2]https://www.geeksforgeeks.org/machine-learning/step-by-step-predictive-analysis-machine-learning/

[3]Empirical Analysis of Computationally Intelligent Technique for Software Risk Prediction

[4]Predictive Analytics for Project Risk Management Using Machine Learning

[5]T. N. H. T. T. Q. N. N. Tran, "Leveraging AI for enhanced software effort estimation: A comprehensive study and framework proposal," arXiv (preprint repository), pp. 284-289, 2024.

[6]M. H. M. T. H. N. D. M. N. A. A. M. A. K. Mahmud, "Software risk prediction: systematic literature review on machine learning techniques," Applied Sciences, vol. 12, 2022.

[7]R. Bhatia, "Predicting Software Development Effort using Machine Learning and Deep Learning," International Journal of Engineering Research & Technology (IJERT), vol. 13, no. 12, 2024.

[8]C. R. M. E. P. G. S. R. Bauskar, "Predictive analytics for project risk management using machine learning," Journal of Data Analysis and Information Processing, vol. 12, no. 4, p. 566–580, 2024.

[9]O. R. Polu, "Machine Learning for Predicting Software Project Failure Risks," IAEME PUBLICATION, vol. 15, no. 4, pp. 950-959, 2024.
[10] **Shabbir et al., 2026 (JCIM)** Empirical Analysis of Computationally Intelligent Technique for Software Risk Prediction
[11]https://www.trueprojectinsight.com/blog/project-office/software-development-life-cycle
[12] Roy, A. (2023). Risk Analysis of Implementing Machine Learning in Construction Projects.
[13] Ibraigheeth, M. and Abu Eid, A.I. (2022). Software Project Risk Assessment Using Machine Learning Approaches. American Journal of Multidisciplinary Research & Development, 4, 35-41.
[14] https://www.geeksforgeeks.org/machine-learning/feature-engineering-scaling-normalization-and-standardization/
[15] https://www.geeksforgeeks.org/machine-learning/how-does-the-classweight-parameter-in-scikit-learn-work/