Adaptive fuzzy inference system-based deep learning model for early-phase software dependability analysis

Deepjyoti Saha[1] Subhashis Chatterjee[2]

(Received 17 February 2025; revised 16 April 2026)

Abstract

Conducting software dependability analysis in the beginning stages of development lowers the likelihood of failures and enhances the overall quality of the system. Some crucial dependability attributes are aging, security, performance, and reliability. The efficiency of traditional dependability analysis and prediction of attributes is limited due to their inability to handle ambiguous, inaccurate, and missing data. This research suggests a fuzzy inference system-based deep neural network model for early-phase software dependability analysis to overcome these difficulties. In order to effectively anticipate dependability attributes and manage the uncertainty of software measurements, a rule formulation algorithm is first developed for the fuzzy inference system model.

DOI:10.21914/anziamproc.v66.19616 Published 2026-05-11, as part of the Proceedings of the 22nd Biennial Computational Techniques and Applications Conference. ISSN 1445-8810. (Print two pages per sheet of paper.)

The estimated dependability attributes are then used to build a deep neural network model that classifies dependable and non-dependable software modules during the early development phase. The deep learning model can reliably classify software modules as trustworthy or not, and it can manage nonlinear interactions of several dependability attributes. Experiments show that the suggested model works better in terms of accuracy, resilience, and flexibility than traditional statistical and machine-learning techniques. This study offers a fresh and clever framework for proactive software dependability evaluation that helps developers to reduce risks early in the software development process.

Contents

1 Introduction

Software has become an essential component of daily life in the current digital era. This drives advancements in various sectors like communication, healthcare, banking, and transportation [22]. As the complexity of these systems continues to increase, it is more important than ever to make sure they are dependable. The capacity of a system to carry out its intended tasks safely, effectively, and consistently under specified conditions is known as software dependability [15]. Software malfunctions can have serious repercussions that range from monetary losses and business interruptions to security lapses and even potentially fatal circumstances [10]. The timing of dependability analysis is one of the most difficult problems in guaranteeing software quality. During later stages of development, such as the testing or deployment phase, many businesses attempt to concentrate on dependability analysis, checking security issues, or performance testing [7, 8, 20]. However, finding and fixing dependability issues at these late stages can be expensive and ineffective. early-phase software dependability analysis helps to mitigate these problems as organizations can identify possible faults or vulnerabilities present in software systems and adopt corrective actions before they escalate into severe problems [4, 3, 23].

To stay competitive and meet deadlines, software companies frequently use legacy code or existing software projects as a basis for the development of new software systems. This method simplifies development, lowers expenses, and frees up teams to concentrate on improving functionality rather than beginning from scratch. However, using legacy software might come with drawbacks like technical debt, security flaws, outdated components, and architectural restrictions, which can affect the reliability and performance

of software systems [21]. Software dependability can also be further compromised by elements like data overflow or underflow, faulty file transfers, and improper code integration. early-phase software dependability analysis, which focuses on significant factors like reliability, security, performability, and aging, is crucial to address these issues [24, 27]. Nevertheless, current research does not offer a comprehensive method that collectively assesses all these elements. This article presents a new framework for software dependability analysis in the early-phases of development for accurately evaluating dependability attributes and identifying dependable and non-dependable software modules based on estimated dependability attributes during the early-phase. This study provides a practical way to improve software resilience and reliability right from the start by combining several dependability metrics.

Software dependability analysis is carried out utilizing a variety of software metrics because software failure data is not readily available in the early stages of software development. These metrics play an important role in evaluating various dependability characteristics. Fenton et al. [11] made early attempts at software reliability estimation by introducing a model based on Bayesian Networks. This strategy was further refined by Chatterjee and Maji [3], who suggested a Bayesian Belief Network-based model for early-phase reliability assessment. Key metrics in software for vulnerability prediction were identified by Filus et al. [12] using a variety of feature selection techniques. More recently, a methodology based on a Deep Learning technique was proposed by Chakraborty et al. [2] to forecast software vulnerabilities in the early stages of software development. Furthermore, early software maintainability prediction has been studied by various researchers [14, 13]. During the early stages of development, machine learning-based models were also employed to identify problems associated with software ageing. An Extreme Machine Learning-based model was suggested by Cotroneo et al. [9] to forecast issues related to software ageing. Most of the existing studies do not account for the uncertainty associated with software metrics when predicting dependability attributes. Fuzzy logic is a powerful tool for handling

uncertainty due to the presence of a rule base in its inference engine to handle uncertainty. Therefore, it can be used to address uncertain software metrics. A Fuzzy Inference System (FIS) model is suggested here to accurately forecast a variety of dependability aspects. To improve the prediction accuracy, a rule-based formulation approach is suggested.

Classifying software modules as dependable or non-dependable during the early-phase of software development is essential for increasing software quality and robustness. Estimated dependability attributes can be used as inputs for classifying the software modules. Classification of software modules during the early-phase reduces resource usage while enhancing software quality. In order to identify dependable and problematic software modules early in the software development process, a deep neural network model is created based on the predicted dependability attributes. Complex linkages and interdependencies between the dependability aspects of the various software modules can be automatically modelled by Deep Neural Networks (DNNs). In the early stages of software development, the suggested rule-based formulation approach for FIS is crucial for estimating dependability attributes and classifying different software modules according to dependability attributes.

Because schedule slippage is still a major problem for software engineers, the main advantage of the current study is that it helps to minimize development costs and make the best use of resources at various stages of the software development life cycle. More importantly, it helps to meet the target time for product release. The primary contributions of this study are as follows.

1. To accurately assess dependability attributes during the early stages of software development, this work proposes a methodology based on a fuzzy inference system.
2. A rule formulation algorithm is developed to estimate dependability attributes within the fuzzy inference framework.
3. A deep neural network model is constructed to classify software modules as trustworthy or untrustworthy in the early development phase.

4. The performance of the proposed model is compared with several alternative approaches to evaluate its effectiveness.

The rest of this study is structured as follows. The key terms needed for early software dependability analysis are defined in Section 2. The suggested approaches for classifying software modules and predicting dependability attributes in the early stages are explained in Section 3. Information on the datasets utilized in the study is given in Section 4. The suggested model is verified and compared with current models in Section 5. The article is finally concluded in Section 6.

2 Software metrics, fuzzy inference system and deep neural network

Section 2.1 provides details on early-phase software metrics. Sections 2.2 and 2.3 provide the conceptual framework of the deep neural network and particulars of the fuzzy inference system.

2.1 Software metrics

Reliability, availability, security, ageing, and performance are the dependability features that can be obtained from early-phase software. The four most crucial software metrics for dependability analysis in the early-phases of software development are covered in the next subsections.

2.1.1 Lines of Code

Lines of code (LOC) are used to assess the size of software components. Kilo lines of code are commonly used to indicate higher software size [19, 26]. Dependability characteristics, including performability, security, reliability, and ageing, can all be impacted by an increase in LOC. Software performance degrades as LOC rises because faults and vulnerabilities are more likely to occur. Therefore, the program becomes less dependable, unsafe, unavailable,

and more vulnerable. Additionally, as LOC increases, software performance deteriorates.

2.1.2 Cyclomatic complexity

The program complexity is evaluated using the cyclomatic complexity (CC) measure, which is computed using a control flow graph [19]. A control flow graph shows every possible way a program could run. The software gets increasingly prone to errors and insecure as CC rises. Software thus becomes hazardous, insecure, and unreliable.

2.1.3 Design complexity

The structural complexity of a software module is measured by its design complexity (DC), which reflects the calling patterns of the modules it interacts with. A more complex module structure results in a higher DC value, and modules involving critical computations tend to have higher DC values [19]. This makes the module more fault-prone and maintenance become difficult. Therefore, higher DC values reduce the reliability, security, and performance of software systems.

2.1.4 Branch count

Branch count (BC) is another crucial design phase software metric that affects software quality. The number of branches in the software program's control flow graph is used to compute BC [5, 6]. A large BC increases the chances of the presence of faults, vulnerabilities, as well as performance issues. This leads to a decrease in reliability and security.

2.2 Fuzzy inference system

A Fuzzy Inference System (FIS) is a popular and useful tool that is used to handle complex as well as real systems, which involves actual processes,

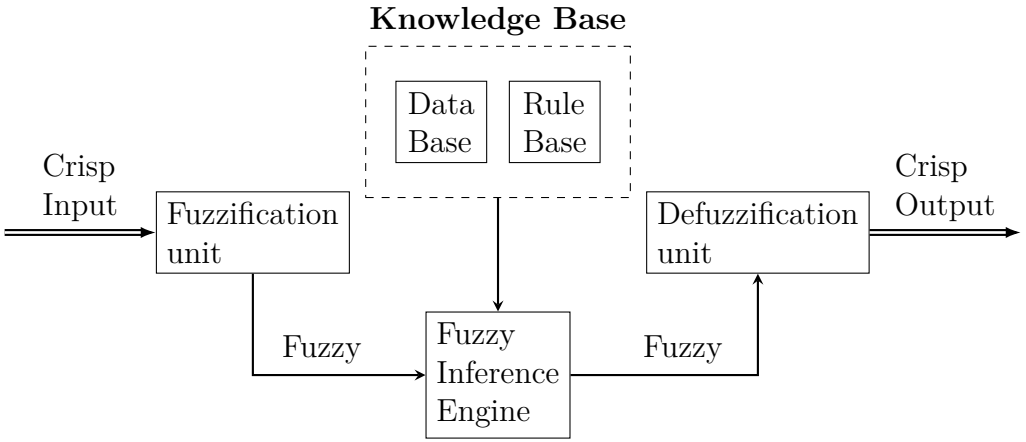

Figure 1: Fuzzy inference system.

uncertainties, and imperfect information and decision-making processes, particularly where uncertainty and imprecision are involved. FIS is widely used in fields like: control systems, decision support, and artificial intelligence, as it can handle uncertainty and approximate reasoning effectively [1, 17]. A FIS allows reasoning with uncertain and imprecise variables [1]. The most important aspect of a FIS is that it represents information in degrees of truth rather than relying solely on binary true or false values.

A FIS typically consists of three main components: a fuzzification unit, an inference engine, and a defuzzification unit. The fuzzification unit transforms crisp inputs into fuzzy values based on predefined membership functions. Different kinds of membership functions are used for fuzzification. There is no fixed rule for selecting the membership functions optimally. They can be chosen with expert's knowledge. To produce fuzzy outputs, the FIS applies fuzzy rules to the transformed fuzzy inputs. These fuzzy outputs are then converted into crisp values by the defuzzification unit for use in decision-making or additional processing. The FIS architecture is seen in Figure 1.

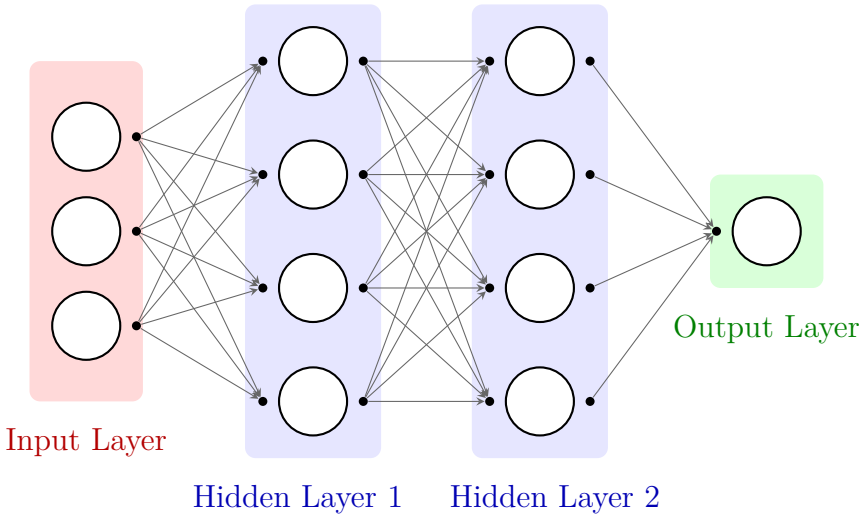

Figure 2: Deep neural network with explicit connection points.

2.3 Deep neural network

A Deep Neural Network (DNN) is a complex artificial neural network model that consists of multiple layers of interconnected nodes or neurons. The complex interconnectivity helps it to model complex patterns and representations from data during training. Each layer of a DNN processes information by applying weighted sums and activation functions during training of the network, and gradually transforming raw input data into higher-level abstractions [28]. The network's depth is determined by the number of hidden layers that exist between the input and output layers. The complex nature of a DNN enables it to learn intricate features and make highly accurate predictions or classifications. The training of a DNN typically adjusts the network's weights based on the error in the output, which enables the model to improve over time. Figure 2 shows the architecture of one DNN model.

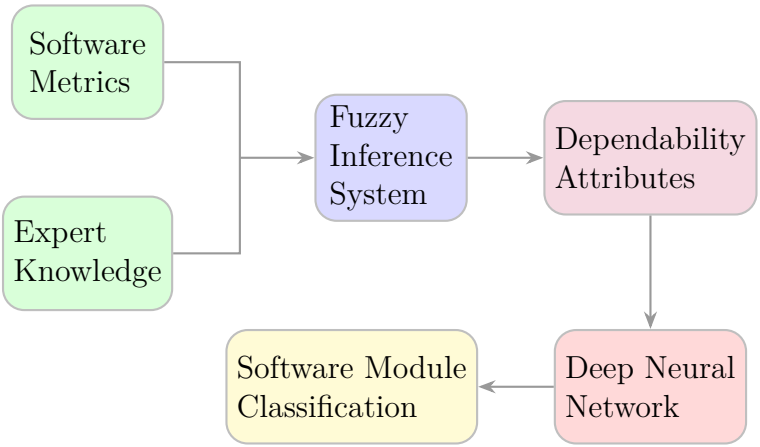

Figure 3: Architecture of proposed methodology.

3 Proposed methodology

Section 3.1 explains the suggested approach for predicting dependability attributes and Section 3.2 discusses the proposed methodology for categorizing dependable and non-dependable software modules in early-stage software development analysis. Figure 3 shows the architecture of the suggested methodology.

During the early stages of software development, software metric values and domain expert knowledge are used to forecast dependability features. Because software metric values can vary in scale based on the metric and measurement instruments used, normalization is crucial to ensure consistency and avoid bias toward specific features or metrics. Various normalization and standardization techniques like Z-score standardization, logarithmic transformation, median-based scaling, or vector normalization techniques, are available. However, the normalization output ranges are not confined to $[0, 1]$. This makes them unsuitable for transforming software metrics into a normalized $[0, 1]$ scale, which is required for the FIS. The FIS operates within a bounded range to define fuzzy membership functions such as “Low”, “Medium”, and “High”,

and hence requires all input variables to be scaled consistently. To overcome this issue, this study employs the min-max normalization technique for the normalization of software metrics. Min–max normalization also maintains the original distribution of the data, avoids negative values, and provides interpretable scaling across metrics with diverse magnitudes, which makes it the most suitable technique for preprocessing software metrics in early-phase software dependability analysis. The min-max normalization approach is mathematically defined as

$$\text{Normalized metric value} = \frac{\text{Original metric value} - \text{Min metric value}}{\text{Max metric value} - \text{Min metric value}}. \quad (1)$$

These normalized values for each software measure are used to forecast dependability attributes in the early stages of software development. Developers gather these metric values and because the process is heavily reliant on human inputs, metrics frequently become ambiguous and imprecise. The fuzzy inference system (FIS) model is one of the most successful techniques in handling data imprecision and uncertainty. The rule base of FIS is the most crucial component for better prediction. Thus, a well-defined rule foundation algorithm is crucial for FIS.

3.1 Fuzzy rule formulation algorithm for dependability attribute prediction

This subsection discusses an algorithm for rule base generation based on expert opinions for FIS. In this FIS, software metrics serve as inputs in the antecedent part, and dependability attributes serve as outputs in the consequent part. Domain expertise is utilized here to develop an efficient rule base for predicting the dependability attributes of various kinds of software systems. Different rules are collected from experts, and these collected rules are used as input for the preparation of rule bases. Algorithm 1 shows the rule formulation algorithm.

Figure 4 depicts the architecture of the proposed rule formulation algorithm for the FIS. The proposed rule formulation algorithm helps the FIS in predicting

Algorithm 1: Algorithm for Rule Formulation of Fuzzy Inference System

Input:

A—number of software metrics, B—number of dependability attributes, L—number of ordinal scales used in software metric value collection, M—number of ordinal scales used in dependability attribute value prediction, S – number of collected rules from experts

Output: Rule base for fuzzy inference system

1: Transform the antecedent part of all expert-defined rules into an L-point ordinal scale.
2: Transform the consequent part of all rules into an M-point ordinal scale.
3: Represent each rule as a point in the feature space.
4: **for** $a = 1$ to S **do**
5: **for** $c = a + 1$ to S **do**
6: Compute Mahalanobis distance:

$$P(a, c) = D(\text{Rule}_a, \text{Rule}_c)$$

7: **end for**
8: **end for**
9: Identify pairs of rules (a, c) with minimum Mahalanobis distance.
10: **for** each pair (a, c) with minimum $D(\text{Rule}_a, \text{Rule}_c)$ **do**
11: Form a new cluster $\text{Cluster}'_a = \text{Rule}_a \cup \text{Rule}_c$.
12: Merge nearby rules into $\text{Cluster}'_a$ if their distance to the cluster centroid is below a threshold.
13: **if** number of elements in $\text{Cluster}'_a \geq 5$ **then**
14: Accept $\text{Cluster}'_a$ as a valid cluster.
15: **else**
16: Continue searching for additional nearest rules.
17: **end if**
18: **end for**
19: Update parameter P based on the final clusters.
20: Generate fuzzy inference rules from the formed clusters.

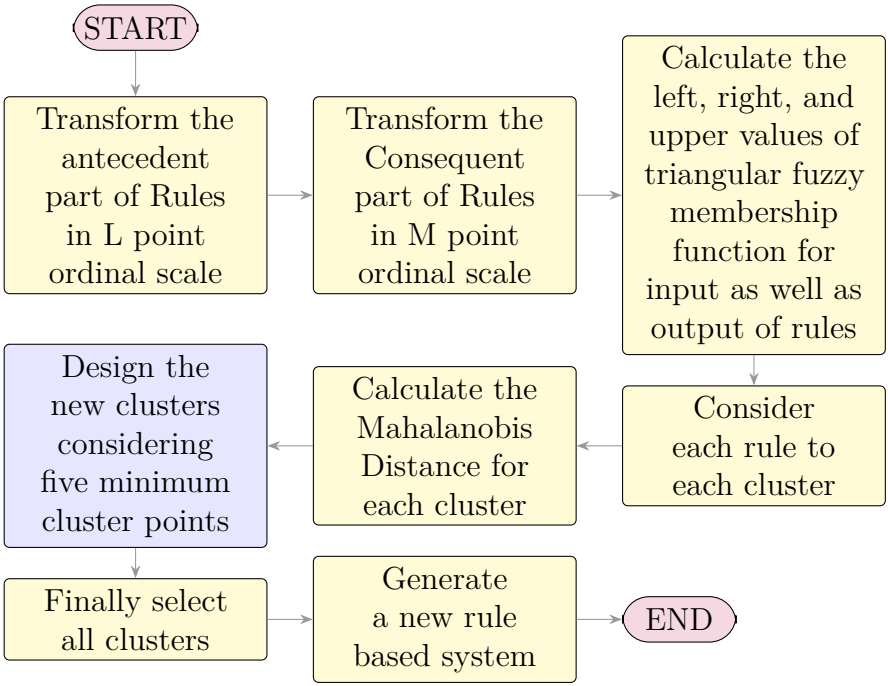

Figure 4: Architecture of the proposed rule formulation algorithm.

various dependability attributes of each software module accurately during the early stages of development. The following subsection describes the categorization technique utilized to conduct software dependability analysis.

3.2 Software module classification via predicted dependability attributes

Classifying software modules according to their dependability index enables software engineers and developers to identify potential weaknesses in the system. Due to various dependability issues like reliability-related faults, vulnerabilities, performance issues, and ageing issues, software becomes non-

Algorithm 2: Deep Neural Network Model for Dependability Prediction

1 **Input:** Datasets containing estimated dependability attributes and target labels (dependable and non-dependable)
2 **Output:** Prediction accuracy
1: Split the dataset into two parts: training and testing sets.
2: Develop the deep neural network (DNN) model.
3: Set the optimizer to **Adam**.
4: Use **Binary Cross-Entropy** as the loss function.
5: Train the model for **50 epochs** with a **batch size of 10**.
6: Evaluate the model using the test dataset.
7: Predict labels for the test data.
8: Compute prediction accuracy and other evaluation metrics.

dependable. Therefore, consideration of these dependability attributes together is important. This section presents a strategy for software module classification during the early-phase. Algorithm 2 gives the dependability training procedure of a deep neural network.

Figure 5 shows the architecture of the suggested classification method. The proposed classification technique contributes to the effective categorization of software modules during early-phase software development.

4 Dataset description

To check the performance of the proposed model, five datasets (kc2, pc1, cm1, kc1, and jm1) are tested. These datasets are sourced from the PROMISE repository [25, 29]. These datasets include key software metrics like: lines of code, McCabe cyclomatic complexity, Halstead metrics, branch count, and others. The details of the datasets are given in Table 1.

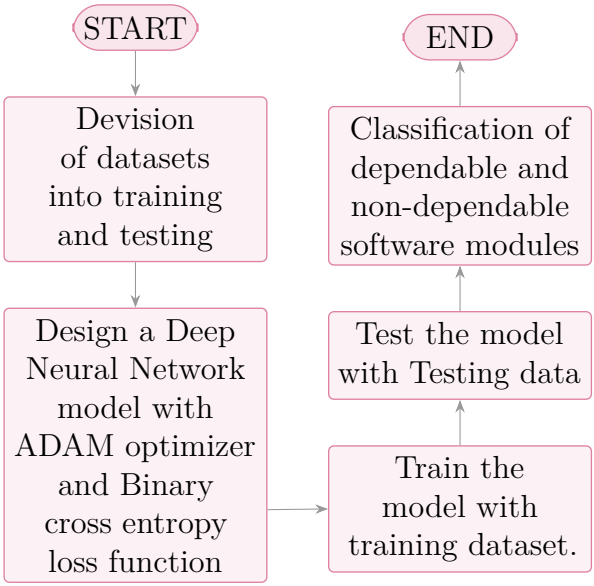

Figure 5: Architecture of the proposed classification algorithm.

Table 1: Details of the datasets for the analysis.

Sl. No.	Name	Module	Code	Description
1	kc2	532	C++	Storage management data
2	pc1	1109	C++	Flight software data
3	cm1	505	C	Spacecraft instrument data processing
4	kc1	2109	C++	Ground-based mission support software
5	jm1	10885	C	Real-time spacecraft monitoring and control system

5 Model implementation and result analysis

Section 5.1 outlines how the suggested model would be implemented. Section 5.2 provides a comparison of performance metrics and concludes with a detailed performance study of the suggested categorization model.

5.1 Proposed model implementation

For the implementation of the proposed methodology, four early-phase software metrics like: lines of code, cyclomatic complexity, design complexity, and branch count are used as inputs, while four dependability attributes like: reliability, performability, security, and aging are considered as output for software dependability analysis during the early development phase. The qualitative data from these software metrics is first normalized, are then used as inputs in a triangle membership function on a five-point ordinal scale. Meanwhile, the proposed FIS-based model estimates dependability attributes on a seven-point ordinal scale. The predicted dependability attributes are then employed for software module classification during the early-phase of development. A DNN model with two hidden layers and ten neurons in each hidden layer is trained to classify software modules. The Adaptable Moment Estimation optimizer is utilized for training model optimization due to its adaptable learning rate, which aids in faster convergence. The binary cross-entropy function is used as the loss function to train the model. To maximize performance, the model is trained for fifty epochs with a batch size of ten.

5.2 Performance comparison and result analysis

The suggested model's efficiency was compared to that of current models using three major comparison criteria: accuracy, F1-score, and Matthew's correlation coefficient (MCC) [6]. The accuracy and F1 scores range from 0 to 1, while the MCC score varies from -1 to 1. Higher scores on these criterion measures suggest a better-fitting model. The logistic regression model [18] and the deep neural network model [16] are used to compare the

Table 2: Performance of different models based on the KC2 dataset.

Model	Accuracy	F1-score	MCC score
Proposed model	0.864	0.500	0.513
Logistic regression model	0.836	0.408	0.380
Deep neural network	0.847	0.491	0.443

Table 3: Performance of different models based on the PC1 dataset.

Model	Accuracy	F1-score	MCC score
Proposed model	0.987	0.857	0.826
Logistic regression model	0.928	0.229	0.192
Deep neural network	0.945	0.231	0.238

effectiveness of the proposed model to existing approaches. Tables 2–6 show the outcomes of the performance evaluations based on five different datasets.

Based on the results tabulated in Tables 2–6, the proposed fuzzy inference-based DNN model consistently outperforms existing models across five benchmark datasets (KC2, PC1, CM1, KC1, and JM1). The findings clearly indi-

Table 4: Performance of different models based on the CM1 dataset.

Model	Accuracy	F1-score	MCC score
Proposed model	0.862	0.527	0.502
Logistic regression model	0.829	0.402	0.371
Deep neural network	0.843	0.485	0.440

Table 5: Performance of different models based on the KC1 dataset.

Model	Accuracy	F1-score	MCC score
Proposed model	0.867	0.533	0.511
Logistic regression model	0.838	0.417	0.382
Deep neural network	0.851	0.497	0.452

Table 6: Performance of different models based on the JM1 dataset.

Model	Accuracy	F1-score	MCC score
Proposed model	0.875	0.549	0.534
Logistic regression model	0.846	0.429	0.397
Deep neural network	0.857	0.502	0.459

cate that the proposed FIS-based framework and rule generation algorithm accurately predict dependability attributes. The DNN-based classification algorithm achieves higher performance in terms of accuracy, F1-score, and MCC score. The rule formulation algorithm developed for the FIS-based model contributes significantly to improve prediction accuracy by refining the fuzzy rule base. Furthermore, the proposed deep neural network (DNN) model, designed for classification, exhibits superior performance compared to traditional approaches. It outperforms both the logistic regression model and the standard DNN, primarily due to the integration of fuzzy logic, which effectively manages uncertainty and imprecision inherent in software metrics. Unlike logistic regression, which assumes linear relationships, the proposed hybrid model captures complex nonlinear patterns, and compared to a conventional DNN, it provides greater interpretability and robustness. By incorporating fuzzification and early-phase dependability attribute estimation, the proposed approach increases predictive accuracy, reduces misclassification errors, and offers a more reliable and generalizable solution for software dependability analysis during the early-phase of software development.

6 Conclusion

In today's fast-paced digital world, software dependability is not just a bonus; it is a must-have. Whether it is safeguarding online banking transactions or keeping self-driving cars running smoothly, reliable software is the backbone of modern technology. A single failure can lead to massive security breaches, financial losses, or even risks to human lives. Businesses and developers need

to focus on dependability from the beginning, using rigorous testing, smart prediction models, and proactive analysis to catch issues before they become major problems. Therefore, a novel, fuzzified deep neural network model is developed for dependability analysis. The proposed rule base formulation algorithm for FIS helps to accurately predict dependability attributes, whereas the DNN model helps classify software modules as dependable and non-dependable during the early development phase. By performing dependability analysis accurately, software companies address various issues related to software systems' reliability, security, and performance. The proposed model helps in building trust, efficiency, and long-term success in an increasingly tech-driven world.

Acknowledgement

The authors are thankful to the reviewers for their beautiful comments, which helped to improve the quality of the paper.

References

[1] A. O. C. Ayres and F. J. Von Zuben. "Multitask learning applied to evolving fuzzy-rule-based predictors". In: *Evolv. Sys.* 12 (2021), pp. 407–422. DOI: 10.1007/s12530-019-09300-w (cit. on p. C200).

[2] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray. "Deep learning based vulnerability detection: Are we there yet?" In: *IEEE Trans. Softw. Eng.* 48 (2022), pp. 3280–3296. DOI: 10.1109/TSE.2021.3087402 (cit. on p. C196).

[3] S. Chatterjee and B. Maji. "A Bayesian belief network based model for predicting software faults in early phase of software development process". In: *Appl. Intel.* 48 (2018), pp. 2214–2228. DOI: 10.1007/s10489-017-1078-x (cit. on pp. C195, C196).

[4] S. Chatterjee and B. Maji. "A fuzzy logic-based model for classifying software modules in order to achieve dependable software". In: *Int. J. Serv. Sci. Man. Eng. Tech.* 11 (2020), pp. 45–57. DOI: 10.4018/IJSSMET.2020100103 (cit. on p. C195).

[5] S. Chatterjee and D. Saha. "IT2F-SEDNN: An interval type-2 fuzzy logic-based stacked ensemble deep learning approach for early phase software dependability analysis". In: *Innov. Syst. Softw. Eng.* 21 (2025), pp. 727–746. DOI: 10.1007/s11334-024-00563-4 (cit. on p. C199).

[6] S. Chatterjee and D. Saha. "Software dependability analysis under neutrosophic environment using optimized Elman recurrent neural network-based classification algorithm and Mahalanobis distance-based ranking algorithm". In: *Ann. Oper. Res.* 340 (2024), pp. 83–115. DOI: 10.1007/s10479-024-05888-8 (cit. on pp. C199, C208).

[7] S. Chatterjee, D. Saha, and A. Sharma. "Multi-upgradation software reliability growth model with dependency of faults under change point and imperfect debugging". In: *J. Softw.: Evol. Proc.* 33.6, e2344 (2021). DOI: 10.1002/smr.2344 (cit. on p. C195).

[8] S. Chatterjee, D. Saha, A. Sharma, and Y. Verma. "Reliability and optimal release time analysis for multi up-gradation software with imperfect debugging and varied testing coverage under the effect of random field environments". In: *Ann. Oper. Res.* 312 (2022), pp. 65–85. DOI: 10.1007/s10479-021-04258-y (cit. on p. C195).

[9] D. Cotroneo, R. Natella, and R. Pietrantuono. "Predicting aging-related bugs using software complexity metrics". In: *Perform. Eval.* 70 (2013), pp. 163–178. DOI: 10.1016/j.peva.2012.09.004 (cit. on p. C196).

[10] N. E. Fenton and N. Ohlsson. "Quantitative analysis of faults and failures in a complex software system". In: *IEEE Trans. Softw. Eng.* 26.8 (2000), pp. 797–814. DOI: 10.1109/32.879815 (cit. on p. C195).

[11] N. Fenton, M. Neil, and D. Marquez. “Using Bayesian networks to predict software defects and reliability”. In: *Proc. Inst. Mech. Eng. O. J. Risk Reliab.* 222 (2008), pp. 701–712. DOI: 10.1243/1748006XJRR161 (cit. on p. C196).

[12] K. Filus, P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe. “Efficient feature selection for static analysis vulnerability prediction”. In: *Sensors* 21, 1133 (2021). DOI: 10.3390/s21041133 (cit. on p. C196).

[13] S. Gupta and A. Chug. “Software maintainability prediction using an enhanced random forest algorithm”. In: *J. Disc. Math. Sci. Crypto.* 23 (2020), pp. 441–449. DOI: 10.1080/09720529.2020.1728898 (cit. on p. C196).

[14] S. Jha, R. Kumar, L. H. Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma, and H. V. Long. “Deep learning approach for software maintainability metrics prediction”. In: *IEEE Access* 7 (2019), pp. 61840–61855. DOI: 10.1109/ACCESS.2019.2913349 (cit. on p. C196).

[15] J. C. Laprie. *Dependability: Basic concepts and terminology.* Dependable computing and fault-tolerant dystems. Springer, 1992. DOI: 10.1007/978-3-7091-9170-5 (cit. on p. C195).

[16] C. Manjula and L. Florence. “Deep neural network based hybrid approach for software defect prediction using software metrics”. In: *Cluster Comput.* 22 (2019), pp. 9847–9863. DOI: 10.1007/s10586-018-1696-z (cit. on p. C208).

[17] P. Martín-Muñoz and F. J. Moreno-Velo. “FuzzyCN2: An algorithm for extracting fuzzy classification rule lists”. In: *2010 IEEE World Congress on Computational Intelligence, WCCI 2010* (2010). URL: https://sci2s.ugr.es/keel/pdf/keel/congreso/fuzzycn2moreno2010.pdf (cit. on p. C200).

[18] G Mauša, T. Galinac Grbac, and B. D. Bašić. "Multivariate logistic regression prediction of fault-proneness in software modules". In: *2012 Proceedings of the 35th International Convention MIPRO*. 2012, pp. 698–703. URL: https://ieeexplore.ieee.org/abstract/document/6240735 (cit. on p. C208).

[19] T. J. McCabe. "A complexity measure". In: *IEEE Transa. Softw. Eng.* SE-2.4 (1976), pp. 308–320. DOI: 10.1109/TSE.1976.233837 (cit. on pp. C198, C199).

[20] C. K. N. C. K. Mohd and F. Shahbodin. "Personalized learning environment: Alpha testing, beta testing and user acceptance test". In: *Procedia Soc. Behav. Sci.* 195 (2015), pp. 837–843. DOI: 10.1016/j.sbspro.2015.06.319 (cit. on p. C195).

[21] A. Mukherjee and D. P. Siewiorek. "Measuring software dependability by robustness benchmarking". In: *IEEE Trans. Softw. Eng.* 23.6 (1997), pp. 366–378. DOI: 10.1109/32.601075 (cit. on p. C196).

[22] H. Pham. *System software reliability*. International series of monographs on physics. Springer, 2006. DOI: 10.1007/1-84628-295-0 (cit. on p. C195).

[23] D. Saha and S. Chatterjee. "Optimized decision tree-based early phase software dependability analysis in uncertain environment". In: *2022 International Interdisciplinary Conference on Mathematics, Engineering and Science (MESIICON)*. 2022, pp. 1–6. DOI: 10.1109/MESIICON55227.2022.10093237 (cit. on p. C195).

[24] J. Sametinger. "Software Security". In: *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. 2013, pp. 216–216. DOI: 10.1109/ECBS.2013.24 (cit. on p. C196).

[25] P. S. Sandhu, S. Khullar, S. Singh, S. K. Bains, M. Kaur, and G. S. "A Study on Early Prediction of Fault Proneness in Software Modules using Genetic Algorithm". In: *World Acad. Sci. Eng. Tech.* 4.12 (2010), pp. 648–653. URL: https://publications.waset.org/10544/a-study-on-early-prediction-of-fault-proneness-in-software-modules-using-genetic-algorithm (cit. on p. C206).

[26] P. Singh, N. R. Pal, S. Verma, and O. P. Vyas. "Fuzzy rule-based approach for software fault prediction". In: *IEEE Trans. Sys. Man. Cybern. Sys.* 47 (2017), pp. 826–837. DOI: 10.1109/TSMC.2016.2521840 (cit. on p. C198).

[27] W. Wang, F. Dumont, N. Niu, and G. Horton. "Detecting software security vulnerabilities via requirements dependency analysis". In: *IEEE Trans. Softw. Eng.* 48 (2022), pp. 1665–1675. DOI: 10.1109/TSE.2020.3030745 (cit. on p. C196).

[28] P. Yu and X. Yan. "Stock price prediction based on deep neural networks". In: *Neural Comput. Appl.* 32 (2020), pp. 1609–1628. DOI: 10.1007/s00521-019-04212-x (cit. on p. C201).

[29] H. Zhang and X. Zhang. "Comments on "Data mining static code attributes to learn defect predictors."" In: *IEEE Trans. Softw. Eng.* 33 (2007), pp. 635–636. DOI: 10.1109/TSE.2007.70706 (cit. on p. C206).

Author addresses

1. **Deepjyoti Saha**, Indian Statistical Institute Kolkata
mailto:sahadeepjyoti01@gmail.com
2. **Subhashis Chatterjee**, Indian Institute of Technology Dhanbad
mailto:subhashis@iitism.ac.in