

# A survey of predicting software reliability using machine learning methods

Shahbaa I. Khaleel, Lumia Faiz Salih

Department of Software, College of Computer Science and Mathematics, Mosul University, Mosul, Iraq

## Article Info

### Article history:

Received Nov 27, 2022

Revised Feb 9, 2023

Accepted Mar 10, 2023

### Keywords:

Artificial intelligence

Deep learning

Machine learning

Prediction

Software reliability

## ABSTRACT

In light of technical and technological progress, software has become an urgent need in every aspect of human life, including the medicine sector and industrial control. Therefore, it is imperative that the software always works flawlessly. The information technology sector has witnessed a rapid expansion in recent years, as software companies can no longer rely only on cost advantages to stay competitive in the market, but programmers must provide reliable and high-quality software, and in order to estimate and predict software reliability using machine learning and deep learning, it was introduced A brief overview of the important scientific contributions to the subject of software reliability, and the researchers' findings of highly efficient methods and techniques for predicting software reliability.

*This is an open access article under the [CC BY-SA](#) license.*



## Corresponding Author:

Shahbaa I. Khaleel

Department of Software, College of Computer Science and Mathematics, Mosul University

Mosul, Iraq

Email: shahbaaibrkh@uomosul.edu.iq

## 1. INTRODUCTION

Companies create smart software to increase software credibility, and thus control failures. Since software in general has real concerns about reliability and maintainability. Researchers have used a variety of machine learning algorithms to find controls for variables that have an impact on most programs [1], [2].

Currently, testing methods are important and most important in determining the usability of software [3]. Software usability is defined as the ability to use the software to its fullest potential without errors within a predetermined period of time [4]. Various techniques are on hand to generate clever programs. Artificial neural networks, fuzzy set theory, approximate set theory, and artificial intelligence are all examples of records retrieval [5]. Some errors formed during error removal and some errors initially present in the data set have the potential to cause the entire system to fail [6].

According to  $A^3$ , the framework shown in Figure 1 of applying program usability, algorithm, and architecture to the reliable work of software without defects. Intelligent software becomes necessary by combining machine learning techniques on the defective company dataset to build reliable models in different dimensions. Based on the early prediction model [7].

Incidents that turn into critical failures as a result of program failure cause financial losses, time losses and information losses [8]. For this reason, errors are handled correctly at the time of release, and they are carefully checked throughout the testing and debugging processes using historical data about software failures to determine the number of test-related errors. Based on the failure history of the application, the best defect handling methods  $m(t)$  and software density function  $\lambda(t)$  are discovered for software reliability models [9].

Machine learning is imperative to flaw detection. It is used in evaluating software program reliability to seem to be for refined variations in how nicely a product works in proper use, and it makes uses a variety of

machine learning techniques to validate a prediction application [10]. Depending on the variety of processing layers via which the facts need to pass, the identify "deep" was once given, and deep studying led to the introduction of neural networks with higher complexity and greater wonderful mastering capabilities, the place the statistical mannequin is produced as output by means of the deep mastering mannequin after making use of a step-by-step non-linear transformation. to an input, and these iterations are made by using the model till the end result is sufficiently accurate [11].



Figure 1.  $A^3$  usability framework for programs

Researchers and software engineers are increasingly integrating deep learning into software engineering (SE) processes. Deep learning benefits SE experts in three main ways: understanding requirements from plain language, generating source code, and predicting software errors. Understanding requirements from plain language is facilitated by deep learning models. These models analyze and extract valuable insights from textual descriptions, enabling a better comprehension of stakeholder needs and expectations.

Deep learning also aids in generating source code. By training on vast amounts of existing code, deep learning models learn patterns, structures, and coding conventions. This enables them to generate code snippets or complete programs based on high-level descriptions or specific requirements, accelerating development efforts and improving productivity.

Predicting software errors is another area where deep learning excels. By analyzing large datasets of code, deep learning models identify patterns indicative of potential bugs or vulnerabilities. This proactive approach allows software engineers to address issues before they become critical problems, enhancing software quality and reliability.

The integration of deep learning into SE processes offers vast potential for advancement. However, it is important to remember that deep learning should be seen as a complement to human expertise rather than a replacement. Domain knowledge, experience, and critical thinking are crucial for ensuring accuracy, reliability, and ethical considerations in applying deep learning techniques in software engineering. By combining the power of deep learning with human intelligence, researchers and software engineers can unlock new possibilities and drive innovation in the SE domain.

The research is organized: The second part clarifies software reliability models, how they are predicted, and their important role in software engineering. The third part dealt with machine learning and its important role in predicting software reliability. The fourth part dealt with previous studies and the findings of researchers in predicting software reliability and using machine learning techniques. As for the fifth part, it dealt with the conclusions reached by the research by reviewing the work of researchers in this field.

## 2. SOFTWARE RELIABILITY MODELS

Over the past few decades, many research on software reliability estimation and prediction have been introduced at conferences, reviewing the improvement of software reliability prediction models is the fundamental effect of this research. These types are based totally on records accumulated at some stage in the checking out segment of the program. The majority of the models that will be put ahead are mathematical features that categorical the relationship between the quantity of blunders observed and the check effort.

The test effort can be calculated using the number of operable test cases, execution time, or calendar time [12]. Models can be used to predict the reliability of the software as properly as the range of defects (or remaining defects) that are no longer detected. Thus, models can assist determine when to end trying out and releasing software. Estimating software reliability is based totally on the primary assumption that as checking out continues greater and greater defects are found. As a result, reliability improves and the variety of defects closing decreases. Hence these models are recognized as software reliability growth models (SRGM) [13].

Non-homogeneous Poisson process (NHPP) models are the most extensively used class of software reliability boom models. As software checking out progresses, NHPP models seem for a heterogeneous method (Poisson) model that excellent matches the error detection sample and use this model to estimate application reliability or residual errors. There is a parameter that represents the predicted complete quantity of software blunders in the majority of these models [14].

Heterogeneous Poisson system models and stochastic system models are the two fundamental classes into which software reliability increase models fall. These two sorts of fashions are extra frequent and frequently used than NHPP models. NHPP models are additionally labeled in accordance to a variety of factors. Figure 2 affords a classification of software reliability increase models [15], [16]. The most important one.

- Test voltage measurement is categorized into models based on the amount of test cases and test duration.
- Exponential and S-shape models, depending on the reliability or average value of the  $m(t)$  function.
- Models with perfect correction and those without are based on different types of correction assumptions.

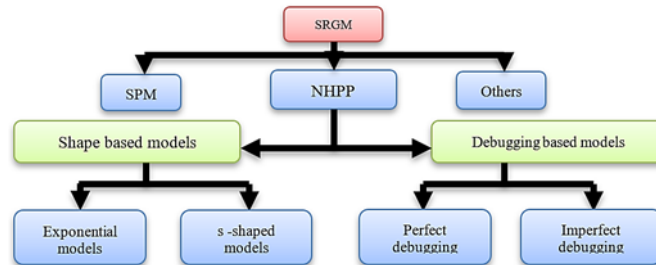


Figure 2. Software reliability growth models according to SRGMs [15]

### 3. MACHINE LEARNING

Neural systems have seen a rush of abundance in the past couple of years and is commonly interconnected across an uncommon array of issue spaces, in disciplines as diverse as finance, solution, design, topography, and materials science. It all started in 1943 when McCulloch and Bates proved that neurons can have two states and that these states can depend on a certain ceiling value. Where they showed for the first time a simulated neuron. Since then, many updated and newer models have been released. The revelation gave McCulloch and Pitt a foothold for intelligent machines [17].

Artificial neural network (ANN) works similarly to a human brain. The probability is that at some point, all people will be the same. Each individual may have made the same judgments in each of the cases. Human nerves may cause one or both of them to react similarly in certain situations, which can be the distinguishing element behind a wide range of human differences [18].

By using machine learning to study data reliability prediction. the methods of artificial intelligince had been studied and employed in software engineering. And that was once carried out thru the usage of the particle swarm optimization (PSO) and crow swarm optimization (CSO) in producing most suitable check instances of the software written with C language in an automated way due to the fact that permits the agency which develops the software to keep time and expenses as properly as making sure the check technique quality, which is estimated via 50% of the product cost [19].

Also using bees swarm to appointment, it to serve software engineering. And that used to be carried out thru the usage of artificial bee’s colony algorithm in resolution of check instances for the software written in C++ language in a computerized way because to allow the business enterprise which develops the software to store time, effort and charges that required for trying out section and regression checking out activity, which is continually evaluated through 50% of the product cost [20]. The estimating in software is used to estimate some necessary and future traits of the software project, such as estimating the developed task effort, and that failure in the application is by and large due to incorrect venture administration practices [21].

Assuming there is a topological network connected by arrows pointing in the right direction. These arrows represent a connection between two neurons and show the direction of information flow. There is a weight for each link, which is an integer representing the signal difference between the two neurons. The structure of the neural network is shown in Figure 3.

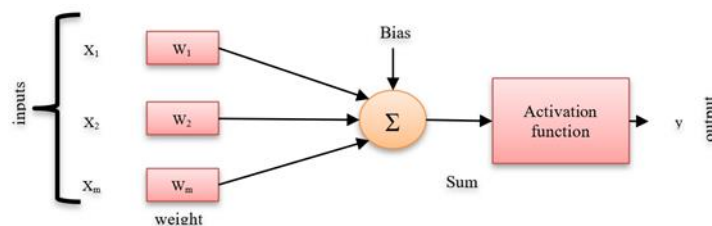


Figure 3. The structure of the neural network

#### 4. DEEP LEARNING

Deep learning methods have recently made significant strides in enhancing research workloads in the field of software engineering. Table 1 provides an overview of the most commonly used deep techniques in this context. Among the widely adopted models are deep belief networks (DBNs), recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory (LSTM). These models have proven effective in a range of software engineering tasks, showcasing the versatility and applicability of deep learning in the field.

Table 1. Combined machine learning and deep learning methods to predict software flaws

Researcher	Techniques	Definition	Advantages	Drawbacks
Wang <i>et al.</i> , 2022 [3]	RNN	RNNs are so named due to the fact they persistently entire the identical venture for each thing in a sequence, with the consequences relying on before calculations.	Processing input of any duration is possible. Model size does not increase with input size. Information from the past is considered during computation.	Sluggish calculation. Accessing knowledge from the past might be challenging. Cannot take into account any potential future changes to the state
Khoshgoftaar <i>et al.</i> 2002 [22]	LSTM	An RNN mannequin regarded as a lengthy non permanent reminiscence (LSTM) community makes use of "forget" gates to get round the vanishing gradient issue.	Retaining knowledge for a very long time.	Training takes longer. More memory is needed to train.
Dam <i>et al.</i> 2018 [23]	CNN	CNNs are a type of deep neural network; at least one of their layer's substitutes convolutions for standard matrix multiplication.	Without any human oversight, it automatically recognizes the crucial characteristics.	need more training data. High computational cost
Wang <i>et al.</i> 2012 [24]	Stacked auto-encoder	The output of each hidden layer is related to the enter of the subsequent hidden layer in a stacked auto encoder, which is a neural community made up of various layers of sparse auto encoders.	Possibility of using transfer learning by using pre-trained layers from another model. Learning can be enabled without labelled inputs.	costly to train in terms of computation incredibly illogical. The underlying mathematics is more difficult. prone to overfitting, however regularization can reduce this
Kamei <i>et al.</i> 2013 [25]	DBN	DBN is a probabilistic, unsupervised deep learning algorithm.	Only a tiny labelled dataset is required. It offers a remedy for the vanishing gradient issue.	It fails to take into account the structural data of programs.
Kumar <i>et al.</i> 2021 [26] Mou <i>et al.</i> 2015 [27]	Logistic regression	Using LR, one can describe data and explain how one dependent binary variable relates to other independent variables.	Easy to implement Very efficient to train	It is unable to mix features to produce new features. Only when input features and output labels have a linear relationship does it function well.
Wang <i>et al.</i> 2018 [28]	Support vector machine (SVM)	A supervised learning model is an SVM. It can be applied to tasks involving regression and classification.	It provides superior prediction results by using alternative kernel functions.	Not appropriate for many software metrics
Manjula and Florence, 2018 [29] Tong <i>et al.</i> 2017 [30]	Decision tree	DT is a decision-support tool that employs a model of decisions and potential outcomes that resembles a tree.	lower computing power Predictive models using tree-based algorithms are more accurate, stable, and comprehensible.	Making a decision tree is difficult.

One of the areas where these deep learning techniques have shown promise is in understanding requirements from plain language. By leveraging DBNs, RNNs, CNNs, and LSTM models, researchers and software engineering experts can analyze textual descriptions and extract meaningful insights to comprehend stakeholder needs and expectations more effectively. This ability to interpret plain language requirements using deep learning methods can greatly improve the accuracy and understanding of project specifications.

Furthermore, these deep learning models have also been successfully applied to generating source code. Through training on vast amounts of existing code, DBNs, RNNs, CNNs, and LSTM models can learn patterns, structures, and coding conventions. This enables them to generate code snippets or even complete programs

based on high-level descriptions or specific requirements. The use of deep learning in code generation has the potential to significantly speed up the development process and boost overall productivity for software engineers.

Traditional neural network models start with random selection of the initial value of the weights. So software flaws cause feature selection to be unstable. By employing a greedy approach, the deep neural network model is able to capture the subtle and reliable aspects of software flaws. Second, traditional neural network models usually make it easier to obtain the local optimal solution. But by using the greedy algorithm, the deep neural network model can find the best overall answer. Compared with previous methods, it can also detect features from software errors more accurately. As a result, the deep neural network model outperforms the regular neural network models in terms of prediction accuracy [31].

## 5. PREVIOUS STUDIES

RNNs have gained significant popularity in addressing problems associated with sequential data. These networks have found extensive application in various domains, including natural language processing, speech recognition, and time series analysis, among others. RNNs excel in handling data sequences due to their ability to retain memory and capture temporal dependencies.

Bai *et al.* [32] has developed a software prediction model based totally on networks (Markov Bayesian), and a technique is proposed to remedy the community model. The researchers assumed that the modern quantity of defects in the application was once normal. This is by and large due to the truth that the regular distribution has many fascinating properties, such as the linear stability, the usage of the (AdaBoosting) algorithm and an accuracy of 82.3%.

Hu *et al.* presented RNNs to describe the interaction between software bug detection and debugging. comparisons with feedforward neural networks and analytical models have been developed. thus, researchers have reached a maximum accuracy of 94.62% [33]. Costa *et al.* presented a method based totally on genetic programming. The use of enhancement methods to enhance overall performance has additionally been proposed. Experiments had been carried out with reliability primarily based on time and take a look at coverage [34].

The result in [35] selected several different forms of SRGM to obtain the self-combining model a self-combination model (ASCM), the second selects several candidate SRGMs to obtain the multiple synthesis model AMCM, and each form of SGRM has been studied, and the results show that ASCM is fairly effective and applicable to improve the estimation and prediction of the performance of the corresponding original SRGM without adding any other factors and assumptions. A multi-combinatorial model (AMCM) is effective and applicable, and also produces better estimation and prediction ability than the neural network-based combinatorial model with an accuracy of 79.63% [35]. Kotaiah and Khan [36] presented a various machine learning strategies or methods to examine software reliability. These methods are, fuzzy method, fuzzy neural strategie, genetic algorithm, Bayesian classification approach, SVM approach, and the self-organization method.

Zhang *et al.* [37] presented main disadvantages of software reliability models based on the basic PSO-SVM evaluation and software reliability prediction properties, some enhanced PSO-SVM metrics have been proposed. The simulation consequences confirmed that in contrast to the classical models, the accelerated model has higher prediction accuracy, higher generalization ability, much less dependence on the range of samples, and it is greater relevant to predict software program reliability through measuring the unit size, which represents the quantity of line codes and the variety of errors, which represents the variety of module defects, and an accuracy of 97.98% was once reached.

The word in [38] describing the inference and statistical prediction of software reliability in the presence of variable information. The Bayesian method was once developed the use of Gaussian strategies and the local occupancy grid map (LOGM) algorithm to estimate the wide variety of application errors over specific time intervals. When the application is assumed to have modified after every time duration and application metrics facts is handy after every update.

Also, Amin *et al.* [39] presented a well-established method to predicting software reliability primarily based on autoregressive integrated moving average (ARIMA) for time sequence as a choice answer to tackle SRGM constraints and supply extra correct dependable prediction. Using real-life datasets on application failures, the accuracy of the proposed strategy used to be evaluated and in contrast to existing, famous approaches. This contrast confirmed that the proposed strategy carried out higher than different ARIMA-based approaches, used to be steady in overall performance and used to be much less high-priced than the SVR approach. An accuracy of 78.80% was once reached.

Zhao *et al.* [40] suggested positive feed back support vector machine (PF-SVR) scheme, the proposed scheme defines the parameters of the SVR model using the full sample data while dynamically adjusting the parameters, and when additional reliability data is received, the parameters of the SVR model are updated using

special equations that include the SVR training model. PF-SVR method provides Improved prediction performance compared to normal SVR performance due to parameter modification. PF-SVR can capture changes in reliability trends by updating adaptive parameters, which makes it convenient for software reliability testing. The MSE scale was used to predict the accuracy of the algorithm and the results were 1.1848, 0.4318 respectively.

While Tyagi and Sharma [41] developed a new component-based software systems (CBSS) mannequin that explains the use of the pathway. Where it has been established that the proposed mannequin the usage of ant colony optimization (ACO) works higher than different models, the reliability of the utility can be estimated by using measuring the time and the opportunity of error. This model gives heuristic component dependency graphs (HCDGs), which assist to estimate CBSS reliability. The HCDGs provide higher reliability estimates than different contemporary techniques with an accuracy of 65.78%.

Roy [42] used some algorithms based on different mathematical approaches such as: fuzzy set theory, different approaches based on time series, wave packet transmission function, which can accurately predict the occurrence of different frequently occurring web errors. The predictive accuracy of the proposed methods is better than a number of current and widely used methods. Moreover, the proposed methods are free from all kinds of unrealistic assumptions such as: the number of errors in the system is limited; Once an error is detected, it is completely removed, the total number of errors detected is proportional to the test time.

While Bhuyan *et al.* [43] used method for predicting software program reliability the use of fuzzy min-max algorithm mixed with recurrent neural technique. An empirical proof has been introduced displaying that the max-min fuzzy algorithm with recurrent method using backpropagation learning offers a correct result. Software reliability prediction has been used to enhance application system manage and acquire excessive software reliability.

Software reliability prediction models proposed by many researchers, where they found some shortcomings as explained in [44]. It has been found that deep learning models are very useful in predicting software errors. RNN-based learning models give better results. Odification in [45] studied the J-M model, the concept of the learn about was once to generalize the proposed risk fee equation by way of including a new structure parameter. The new customary risk ratio method is very bendy to accommodate all varieties of time-dependent conduct. can provide a range of SRGMs that can be used with much less effort and time in any methods decision study.

The two researchers Tamura and Yamada in [46] have proposed a method for selecting the optimal program reliability model based on deep learning. Many numerical examples of software reliability assessment are presented in actual software projects. Where the optimum release time and the expected total cost of the program were discussed in terms of model selection based on deep learning, the proposed method based on deep learning showed a better potential than that based on neural network.

While Xu *et al.* [47] used an approach multi-layered heterogeneous dynamic particle swarm optimization-back propagation (MHPSO-BP) for software reliability prediction that is based on a more effective multi-layer heterogeneous PSO neural network BP. This approach uses an attractor to optimize the pace replace equation for the particle and sets the demography of the particle swarm to a hierarchical structure. The particle swarm technique has been optimized, and the statistics interaction between particles has been improved. Then, the optimised PSO was once applied to raise the neural network weight and threshold BP during the experiment, the software reliability prediction test was run using dataset from the NASA metrics data program (NASAMDP). The results showed that the suggested method has better prediction performance overall than the typical neural for back propagation via 92%.

While Wang [48] analyzed the necessities for prediction of software program reliability mannequin and contrast system, describing the standard shape of the system, the precise unit features and database design. Where JavaScript, HyperText Markup Language (HTML) and different applied sciences have been used to whole the diagram of the software reliability contrast machine and evaluation of the hierarchical shape of training and essential software packages. And the check consequences exhibit that the software reliability predictive machine can meet the commercial enterprise requirements, and with an accuracy of 94.01%.

The researchers Pattnaik and Ray [49] discussed the reliability of existing software, estimation models at different stages of the software development process, and metrics used for software reliability at different levels ie, code level and architectural level. Various models have been represented for reliability analysis. Most of them are derived analytically from assumptions. The limitations of prediction models as well as architectural models are also discussed. The effect of failure data on software reliability prediction has been observed, and it has been analytically observed that the exponential distribution plays an important role in reliability since it has a constant failure rate. Finally, some familiar tools for measuring the expectation and estimation of software reliability are discussed.

While Barack and Huang [50] studied cellular utility reliability evaluation and prediction the usage of frequent software reliability increase models SRGMs, the four software reliability models are used to consider

the reliability of an open supply cellular utility through examining computer virus reports. Experiments have validated that it is viable to use SRGM with fault records got from error reviews to consider and predict software program reliability in cell applications. The consequences of the find out about allow software program builders and testers to evaluate and predict the reliability of cellular software program functions.

The researchers Sahu and Srivastava [51] have studied a number of already developed reliability growth models (RGM) and used them at different stages of development respectively. This was found in the study that there is no reliable prediction model that can be used during the software development process. The researchers provide suggestions for developers to develop and describe a reliable prediction model that can be used with every stage of development.

Also, Gandhi *et al.* [52] presented a high quality algorithm that can be used to predict the reliability of the program. The proposed algorithm is applied the usage of a hybrid strategy referred to as neuro-fuzzy inference system and it has additionally been utilized to the take a look at data. After checking out and coaching real-time records with reliability prediction in phrases of imply relative error and suggest absolute relative error as 0.0060 and 0.0121 respectively. The consequences exhibit that the proposed algorithm predicts captivating outcomes in phrases of the absolute imply relative error as properly as the imply relative error in contrast to different current models that justify the dependable prediction of the proposed model. Thus, this new technological know-how goals to make this model as easy as viable to enhance software reliability.

Kushwah and Sharma [53] by examining the nature of the labour in the software process, researchers explored the prediction of software failure. The research found that the software program dependability prediction models put forth by numerous researchers had some flaws and didn't work in all test conditions. Additionally, assessing the trustworthiness of software programmes is no longer an actual science. Soft computing techniques including neural networks, fuzzy logic, genetic algorithms, genetic programming, swarm intelligence, and bayesian networks, among others, are of utmost significance. While the use of modern light computing techniques in software for dependability modelling is stressed.

While San *et al.* [54] presented a new technique for software program reliability modeling known as deep projects software reliability growth model deep cross-project software reliability growth model and this approach is a cross-project forecasting approach that makes use of the elements of previous tasks records via challenge similarity. Specifically, the proposed technique applies block-based mission resolution of coaching and modeling statistics supply by using a deep mastering method. Experimental find out about outcomes that encompass 15 actual E-Seikatsu datasets and eleven open supply software program datasets exhibit that DC-SRGM can greater precisely describe the reliability of ongoing improvement tasks than the contemporary traditional SRGM and LSTM models.

Ali *et al.* [55] presented a reliability prediction mannequin that enhances scalability by using introducing an algorithmic mechanism TypeScript state machine. In addition, the proposed method helps modeling the nature of concurrent functions by way of adapting the formal statistical distribution in the direction of the situation set. The proposed method was once evaluated the use of sensor-based case studies. The experimental outcomes confirmed the effectiveness of the proposed method from the factor of view of lowering the computational price in contrast to comparable models. This discount is the most important parameter to enhance scalability. In addition, the introduced work can allow gadget builders to be aware of the load their device will be dependable with the aid of watching the reliability fee in many running situations.

After reviewing these studies, they are summarized in Table 2 (see in Appendix). It shows the database used, whether it was previously stored data or real-time data. To mentioning the scale used to determine the quality and accuracy of the technology used to predict the reliability of the software.

Table 2 provides evidence that the utilization of machine learning techniques yields satisfactory accuracy when assessing the reliability of software programs. The high accuracy rates achieved can be attributed to the quality of the technology employed, regardless of whether the database is extensive or of moderate size. This implies that machine learning algorithms have the capability to effectively determine the reliability of programs, regardless of the scale of the database being analyzed.

## 6. CONCLUSION

Using deep learning is the best solution for ensuring software reliability, according to previous discussions. Ensuring software reliability has become a serious concern due to the increasing size and complexity of the current software. Anticipating potential code defects in software applications can be considered a useful way to increase software reliability since it can significantly reduce software maintenance work. A flaw prediction framework that uses deep learning algorithms to automatically generate features from source code while preserving semantic and structural information has the greatest promise. Moreover, our survey confirms the feasibility of deep learning methods for programming and its important role in using it to predict software reliability.

## APPENDIX

Table 2. Summarizes the relevant works

Researchers	Algorithm	Dataset	Metric	Percentage
Wang <i>et al.</i> 2018 [28]	AdaBoosting	ASCM	Accuracy	82.3%
Hu <i>et al.</i> 2007 [33]	Genetic Algorithm	Real-time command and control application consisting of 21,700 assembly instructions	Accuracy	67.24%
		Flight dynamic application consisting of 10,000 lines of code	Accuracy	80.00%
		Flight dynamic application consisting of 22,500 lines of code	Accuracy	89.61%
		Flight dynamic application consisting of 38,500 lines of code	Accuracy	94.62%
Costa <i>et al.</i> 2007 [34]	Markov chain Monte Carlo	Real dataset	Accuracy	-
Li <i>et al.</i> 2011 [35]	SVM with Genetic Algorithm	SVR dataset	Accuracy	79.63%
Kotaiah and Khan, 2012 [36]	SVR	Sys1 Sys3	Accuracy	-
Zhang <i>et al.</i> 2013 [37]	PSO-SVM PSO-LSSVM BP	collected during testing phase	Accuracy	85.5% 89.46% 97.98%
Amin <i>et al.</i> 2013 [39]	SRGMs	Sys40	Accuracy	78.80%
Graves 2013 [56]	LSTM	Hutter prize	Accuracy	79.64%
Zhao <i>et al.</i> 2013 [40]	SVR	actual error data	MSE	1.1848
	PF-SVR	actual error data	MSE	0.4318
Tyagi and Sharma [41]	Ant Colony Optimization Reliability	collected during testing phase		65.78%
Cho <i>et al.</i> 2014 [57]	RNN	UNK	Accuracy	92.01%
Tian and Noore 2015 [58]	GA	John Musa	Accuracy	98.57%
Roy 2015 [42]	fuzzy forecasting	HTTP logs	Accuracy	95.2%
Bhuyan <i>et al.</i> 2016 [43]	Max-min	-	AE	3.0019
Al Turk and Alsolami, 2016 [45]	Software Reliability Growth Models	JM	-	-
Tamura and Yamada, 2016 [46]	Neural network	Actual dataset	Accuracy	67%
	Deep Learning	Actual dataset	Accuracy	83%
Xu <i>et al.</i> 2017 [47]	MHPSO-BP	JM1	Accuracy	92.00%
Wang <i>et al.</i> 2018 [48]	JM GO MBN	Space consists of 9564 lines of C code	Accuracy	69.84% 85.99% 94.01%
Pattnaik and Ray, 2020 [49]	SRGMs	available failure data	Accuracy	98.6%
Barack and Huang, 2020 [50]	SRGMs	-	Accuracy	98.6%
Sahu and Srivastava, 2020 [51]	RGM	Many online datasets	-	-
Gandhi <i>et al.</i> 2020 [52]	Neuro-Fuzzy Inference System	-	MRE	0.0121
San <i>et al.</i> 2021 [54]	DC-SRGM LSTM Logistic	online datasets online datasets online datasets	AE AE AE	0.110 0.146 0.220
Ali <i>et al.</i> 2022 [55]	s-TS + FSMS	-	Failure Propability	0.01

## ACKNOWLEDGEMENTS

Authors would like to thank the University of Mosul in Iraq for providing moral support.

## REFERENCES

- [1] N. Gupta, A. Rana, and S. Gupta, "Fitness for solving SMCP using evolutionary algorithm," *IOP Conference Series: Materials Science and Engineering*, vol. 1099, no. 1, p. 012041, Mar. 2021, doi: 10.1088/1757-899x/1099/1/012041.
- [2] L. L. Silva, M. T. Valente, A. M. De Maia, and N. Anquetil, "Developers' perception of co-change patterns: An empirical study," in *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*, Sep. 2015, pp. 21–30, doi: 10.1109/ICSM.2015.7332448.
- [3] J. Wang, C. Zhang, and J. Yang, "Software reliability model of open source software based on the decreasing trend of fault introduction," *PLoS ONE*, vol. 17, no. 5 May, p. e0267171, May 2022, doi: 10.1371/journal.pone.0267171.
- [4] R. Arora and A. G. Aggarwal, "Testing effort based software reliability assessment incorporating FRF and change point," *Yugoslav Journal of Operations Research*, vol. 30, no. 2, pp. 271–286, 2020, doi: 10.2298/YJOR190315022A.
- [5] N. K. Kahlon, K. K. Chahal, and S. B. Narang, "Managing QoS degradation of component web services in a dynamic environment,"






- International Journal on Semantic Web and Information Systems*, vol. 14, no. 2, pp. 162–190, Apr. 2018, doi: 10.4018/IJSWIS.2018040108.
- [6] K. L. Peng and C. Y. Huang, “Reliability analysis of on-demand service-based software systems considering failure dependencies,” *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 423–435, May 2017, doi: 10.1109/TSC.2015.2473843.
- [7] M. Banga, A. Bansal, and A. Singh, “Implementation of machine learning techniques in software reliability: A framework,” in *2019 International Conference on Automation, Computational and Technology Management, ICACTM 2019*, Apr. 2019, pp. 241–245, doi: 10.1109/ICACTM.2019.8776830.
- [8] M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang, “File-level defect prediction: Unsupervised vs. Supervised models,” in *International Symposium on Empirical Software Engineering and Measurement*, Nov. 2017, vol. 2017-Novem, pp. 344–353, doi: 10.1109/ESEM.2017.48.
- [9] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, “Preliminary comparison of techniques for dealing with imbalance in software defect prediction,” May 2014, doi: 10.1145/2601248.2601294.
- [10] R. B. Bahaweres, F. Agustian, I. Hermadi, A. I. Suroso, and Y. Arkeman, “Software defect prediction using neural network based smote,” in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Oct. 2020, vol. 2020-October, pp. 71–76, doi: 10.23919/EECSI50503.2020.9251874.
- [11] L. Qiao, X. Li, Q. Umer, and P. Guo, “Deep learning based software defect prediction,” *Neurocomputing*, vol. 385, pp. 100–110, Apr. 2020, doi: 10.1016/j.neucom.2019.11.067.
- [12] G. Gayathry and R. Thirumalai Selvi, “Classification of software reliability models to improve the reliability of software,” *Indian Journal of Science and Technology*, vol. 8, no. 29, Nov. 2015, doi: 10.17485/ijst/2015/v8i29/85287.
- [13] P. K. Kapur, S. Anand, K. Yadav, and J. Singh, “A unified scheme for developing software reliability growth models using stochastic differential equations,” *International Journal of Operational Research*, vol. 15, no. 1, pp. 48–63, 2012, doi: 10.1504/IJOR.2012.048291.
- [14] J. Iqbal, “Analysis of some software reliability growth models with learning effects,” *International Journal of Mathematical Sciences and Computing*, vol. 2, no. 3, pp. 58–70, Jul. 2016, doi: 10.5815/ijmsc.2016.03.06.
- [15] B. John, “A brief review of software reliability prediction models,” *International Journal for Research in Applied Science and Engineering Technology*, vol. V, no. IV, pp. 990–997, Apr. 2017, doi: 10.22214/ijraset.2017.4180.
- [16] R. B. Karaomer, “Comparison of non-homogeneous poisson process software reliability models in web applications,” *AJIT-e Online Academic Journal of Information Technology*, pp. 7–28, Aug. 2016, doi: 10.5824/1309-1581.2016.3.001.x.
- [17] G. K. Mohan, N. Yoshitha, M. L. N. Lavanya, and A. K. Priya, “Assessment and analysis of software reliability using machine learning techniques,” *International Journal of Engineering & Technology*, vol. 7, no. 2.32, p. 201, May 2018, doi: 10.14419/ijet.v7i2.32.15567.
- [18] S. Omri and C. Sinz, “Machine learning techniques for software quality assurance: A survey,” *Computer Science > Software Engineering*, 2021, doi: 10.48550/arXiv.2104.14056.
- [19] S. I. Khaleel and A. Al Thanoon, “Design a tool for generating test cases using swarm intelligence,” *AL-Rafidain Journal of Computer Sciences and Mathematics*, vol. 10, no. 1, pp. 421–444, Mar. 2013, doi: 10.33899/csmj.2013.163468.
- [20] S. I. Khaleel and R. Khaled, “Selection and prioritization of test cases by using bees colony,” *AL-Rafidain Journal of Computer Sciences and Mathematics*, vol. 11, no. 1, pp. 179–201, Jul. 2014, doi: 10.33899/csmj.2014.163746.
- [21] S. I. Khaleel, “Designing a tool to estimate software projects based on the swarm intelligence,” *International Journal of Intelligent Engineering and Systems*, vol. 14, no. 4, pp. 524–538, Aug. 2021, doi: 10.22266/ijies2021.0831.46.
- [22] T. M. Khoshgoftar, E. B. Allen, and J. Deng, “Using regression trees to classify fault-prone software modules,” *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 455–462, Dec. 2002, doi: 10.1109/TR.2002.804488.
- [23] H. K. Dam *et al.*, “A deep tree-based model for software defect prediction,” *Computer Science > Software Engineering*, 2018, doi: 10.48550/arXiv.1802.00921.
- [24] J. Wang, B. Shen, and Y. Chen, “Compressed C4.5 models for software defect prediction,” in *Proceedings-International Conference on Quality Software*, Aug. 2012, pp. 13–16, doi: 10.1109/QSIC.2012.19.
- [25] Y. Kamei *et al.*, “A large-scale empirical study of just-in-time quality assurance,” *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, Jun. 2013, doi: 10.1109/TSE.2012.70.
- [26] P. S. Kumar, H. S. Behera, J. Nayak, and B. Naik, “Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature,” *Innovations in Systems and Software Engineering*, vol. 17, no. 4, pp. 355–379, May 2021, doi: 10.1007/s11334-021-00399-2.
- [27] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, “Convolutional neural networks over tree structures for programming language processing,” *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, vol. 30, no. 1, pp. 1287–1293, Feb. 2016, doi: 10.1609/aaai.v30i1.10139.
- [28] S. Wang, T. Liu, J. Nam, and L. Tan, “Deep semantic feature learning for software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1267–1293, Dec. 2020, doi: 10.1109/TSE.2018.2877612.
- [29] C. Manjula and L. Florence, “Deep neural network based hybrid approach for software defect prediction using software metrics,” *Cluster Computing*, vol. 22, no. S4, pp. 9847–9863, Jan. 2019, doi: 10.1007/s10586-018-1696-z.
- [30] H. Tong, B. Liu, and S. Wang, “Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning,” *Information and Software Technology*, vol. 96, pp. 94–111, Apr. 2018, doi: 10.1016/j.infsof.2017.11.008.
- [31] R. Khan and M. Mahajan, “A review on automatic testing of deep learning systems,” *Journal of Xi'an University of Architecture & Technology*, vol. XII, no. Iv, pp. 1822–1832, 2020.
- [32] C. G. Bai, Q. P. Hu, M. Xie, and S. H. Ng, “Software failure prediction based on a Markov Bayesian network model,” *Journal of Systems and Software*, vol. 74, no. 3, pp. 275–282, Feb. 2005, doi: 10.1016/j.jss.2004.02.028.
- [33] Q. P. Hu, M. Xie, S. H. Ng, and G. Levitin, “Robust recurrent neural network modeling for software fault detection and correction prediction,” *Reliability Engineering and System Safety*, vol. 92, no. 3, pp. 332–340, Mar. 2007, doi: 10.1016/j.res.2006.04.007.
- [34] E. O. Costa, G. A. de Souza, A. T. R. Pozo, and S. R. Vergilio, “Exploring genetic programming and boosting techniques to model software reliability,” *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 422–434, Sep. 2007, doi: 10.1109/TR.2007.903269.
- [35] H. Li, M. Zeng, M. Lu, X. Hu, and Z. Li, “Adaboosting-based dynamic weighted combination of software reliability growth models,” *Quality and Reliability Engineering International*, vol. 28, no. 1, pp. 67–84, May 2012, doi: 10.1002/qre.1216.
- [36] B. Kotaiah and R. A. Khan, “A survey on software reliability assessment by using different machine learning techniques,” *International Journal of Scientific & Engineering Research*, vol. 3, no. 6, pp. 1–7, 2012.
- [37] X. Zhang, J. Yang, S. Du, and S. Huang, “A new method on software reliability prediction,” *Mathematical Problems in Engineering*, vol. 2013, pp. 1–8, 2013, doi: 10.1155/2013/385372.
- [38] N. Torrado, M. P. Wiper, and R. E. Lillo, “Software reliability modeling with software metrics data via gaussian processes,” *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1179–1186, Aug. 2013, doi: 10.1109/TSE.2012.87.




- [39] A. Amin, L. Grunske, and A. Colman, "An approach to software reliability prediction based on time series modeling," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1923–1932, Jul. 2013, doi: 10.1016/j.jss.2013.03.045.
- [40] W. Zhao, T. Tao, Z. Ding, and E. Zio, "A dynamic particle filter-support vector regression method for reliability prediction," *Reliability Engineering and System Safety*, vol. 119, pp. 109–116, Nov. 2013, doi: 10.1016/j.res.2013.05.021.
- [41] K. Tyagi and A. Sharma, "A heuristic model for estimating component-based software system reliability using ant colony optimization," *World Applied Sciences Journal*, vol. 31, no. 11, pp. 1983–1991, 2014.
- [42] A. Roy, "Web software reliability analysis using various mathematical approaches," no. March, 2015.
- [43] M. K. Bhuyan, D. P. Mohapatra, and S. Sethi, "Software reliability prediction using fuzzy min-max algorithm and recurrent neural network approach," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 4, pp. 1929–1938, Aug. 2016, doi: 10.11591/ijece.v6i4.9991.
- [44] N. Rastogi, "Survey on software reliability prediction using soft computing," *Int. J. Comput. Eng. Technol.*, vol. 9, no. 4, pp. 212–216, 2018.
- [45] L. Al turk and E. Alsolami, "Real data application and analysis for evaluating several sub-models of generalized jelinski-moranda formula," *International Journal of Development Research*, vol. 6, no. 10, pp. 9651–9656, 2016.
- [46] Y. Tamura and S. Yamada, "Software reliability model selection based on deep learning with application to the optimal release problem," *Journal of Industrial Engineering and Management Science*, vol. 2016, no. 1, pp. 43–58, May 2016, doi: 10.13052/jiems2446-1822.2016.003.
- [47] D. Xu, S. Ji, Y. Meng, and Z. Zhang, "A software reliability prediction algorithm based on MHP SO - BP neural network," 2017, doi: 10.2991/gcmce-17.2017.10.
- [48] R. Wang, "Research on software reliability model prediction and evaluation system," *MATEC Web of Conferences*, vol. 228, p. 1013, 2018, doi: 10.1051/mateconf/201822801013.
- [49] S. C. Pattnaik\* and M. Ray, "Software reliability prediction and estimation," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 8, pp. 855–869, Jun. 2020, doi: 10.35940/ijitee.h6329.069820.
- [50] O. Barack and L. Huang, "Assessment and prediction of software reliability in mobile applications," *Journal of Software Engineering and Applications*, vol. 13, no. 09, pp. 179–190, 2020, doi: 10.4236/jsea.2020.139012.
- [51] K. Sahu and R. K. Srivastava, "Needs and importance of reliability prediction: An industrial perspective," *Information Sciences Letters*, vol. 9, no. 1, pp. 33–37, Jan. 2020, doi: 10.18576/isl/090105.
- [52] P. Gandhi, M. Z. Khan, R. K. Sharma, O. H. Alhazmi, S. Bhatia, and C. Chakraborty, "Software reliability assessment using hybrid neuro-fuzzy model," *Computer Systems Science and Engineering*, vol. 41, no. 3, pp. 891–902, 2022, doi: 10.32604/csse.2022.019943.
- [53] R. S. Parmanand Kushwah, "Soft computing techniques for software reliability," *International Journal of Scientific Engineering and Research (IJSER) ISSN (Online): 2347-3878 Impact Factor (2020): 6.733*, vol. 9, no. 7, pp. 133–140, 2020.
- [54] K. K. San, H. Washizaki, Y. Fukazawa, K. Honda, M. Taga, and A. Matsuzaki, "Deep cross-project software reliability growth model using project similarity-based clustering," *Mathematics*, vol. 9, no. 22, 2021, doi: 10.3390/math9222945.
- [55] A. Ali *et al.*, "Design-time reliability prediction model for component-based software systems," *Sensors*, vol. 22, no. 7, p. 2812, Apr. 2022, doi: 10.3390/s22072812.
- [56] A. Graves, "Generating sequences with recurrent neural networks," *Computer Science > Neural and Evolutionary Computing*, 2013, doi: 10.48550/arXiv.1308.0850.
- [57] R. Robins-Browne, S. Cianciosi, and J. G. Morris, "Evaluation of different techniques for detection of virulence in *Yersinia enterocolitica*," *Journal of Clinical Microbiology*, vol. 28, no. 9, p. 2159, Sep. 1990, doi: 10.1128/jcm.28.9.2159-1990.
- [58] L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," *Reliability Engineering and System Safety*, vol. 87, no. 1, pp. 45–51, Jan. 2005, doi: 10.1016/j.res.2004.03.028.

## BIOGRAPHIES OF AUTHORS



**Shahbaa I. Khaleel**    was born in Mosul, Nineveh, Iraq, she received the B.S., M.Sc. and Ph.D. degrees in computer science from Mosul University, in 1994, 2000 and 2006, respectively, assistant prof. since 2011, and finally, prof. degree on 2021. From 2000 to 2023, she was taught computer science, software engineering and techniques in the College of Computer Sciences and Mathematics, University of Mosul. She has research in a field computer science, software engineering, intelligent technologies. She can be contacted at shahbaaibrkh@uomosul.edu.iq.



**Lumia Faiz**    was born in Mosul, Nineveh, Iraq, she received the B.S., degree in software engineering from the Mosul University, in 2007, and she study now Master Degree in Software Department, College of Computer Science and Mathematics, Mosul University, Iraq. She can be contacted at email: lumia.21csp2@student.uomosul.edu.iq.