PAPER

MuSRGM: A Genetic Algorithm-Based Dynamic Combinatorial Deep Learning Model for Software Reliability Engineering

Ning FU[†a)], **Duksan RYU**[†b)], ***and*** **Suntae KIM**[†c)], ***Nonmembers***

SUMMARY In the software testing phase, software reliability growth models (SRGMs) are commonly used to evaluate the reliability of software systems. Traditional SRGMs are restricted by their assumption of a continuous growth pattern for the failure detection rate (FDR) throughout the testing phase. However, the assumption is compromised by Change-Point phenomena, where FDR fluctuations stem from variations in testing personnel or procedural modifications, leading to reduced prediction accuracy and compromised software reliability assessments. Therefore, the objective of this study is to improve software reliability prediction using a novel approach that combines genetic algorithm (GA) and deep learning-based SRGMs to account for the Change-point phenomenon. The proposed approach uses a GA to dynamically combine activation functions from various deep learning-based SRGMs into a new mutated SRGM called MuSRGM. The MuSRGM captures the advantages of both concave and S-shaped SRGMs and is better suited to capture the change-point phenomenon during testing and more accurately reflect actual testing situations. Additionally, failure data is treated as a time series and analyzed using a combination of Long Short-Term Memory (LSTM) and Attention mechanisms. To assess the performance of MuSRGM, we conducted experiments on three distinct failure datasets. The results indicate that MuSRGM outperformed the baseline method, exhibiting low prediction error (MSE) on all three datasets. Furthermore, MuSRGM demonstrated remarkable generalization ability on these datasets, remaining unaffected by uneven data distribution. Therefore, MuSRGM represents a highly promising advanced solution that can provide increased accuracy and applicability for software reliability assessment during the testing phase.

key words: *genetic algorithm, software reliability growth models, LSTM, attention, deep-learning*

1. Introduction

Reducing the development time and cost while maintaining software quality is a challenge faced by all software companies. To address this issue, SRGMs have gained popularity as a measure of software quality [1]. These models use statistics to establish a correlation between failure data and a known function [2], such as an exponential function [3]. If the correlation is confirmed, the function can be used to predict the number of future failures that will occur in the software system.

Many SRGMs have been developed and applied during software testing to assess software reliability [4]. The two primary categories of SRGMs are the concave model and the S-shaped model [5], [6], as shown in Fig. 1. The concave model is based on the assumption that failures are more readily identified and resolved during the initial testing phases. Consequently, FDR experienced a rapid decline at the beginning, gradually tapering off over time and forming a concave trend [7]. In contrast, the S-shaped model suggests that FDR undergoes a gradual rise in the initial stages of testing [8], [9]. This initial phase often reflects the learning process of the testing team and the complexity of uncovering less obvious faults. As testing progresses and the team's understanding deepens, the FDR is anticipated to continue its upward trend [10]. When using SRGMs, one of these categories must be selected and associated with the failure data [11]. Traditional SRGMs assume that the FDR follows a continuous growth pattern, such as constant, exponential, or power-law, throughout the testing phase [12]. However, this assumption is invalid if there are unexpected changes in testers or testing strategies, known as change-points. The change in FDR due to change-points affects the accuracy of software reliability assessments using traditional SRGMs.

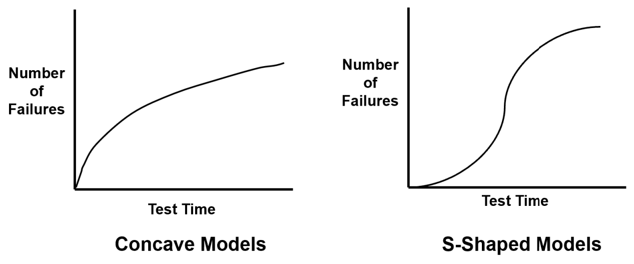


Fig. 1 Concave and S-shaped model

Researchers have explored different strategies to enhance the predictive performance of SRGMs. One approach involves considering change-points to improve fault prediction accuracy and achieve satisfactory forecasting precision. Jing Zhao et al. [13] and Vikas Dhaka with Nidhi Nijhawan [14] have incorporated change-points and environmental factors, significantly enhancing accuracy. Nevertheless, these methods still rely on non-homogeneous Poisson process (NHPP) functions, leading to a noticeable gap in predictive accuracy compared to artificial intelligence models. Another avenue of research focuses on leveraging deep learning techniques to enhance software reliability prediction [15]. Wu and Huan [16] proposed a method to transform traditional SRGMs into deep-learning models, yielding promising results. However, their approach only wraps the traditional SRGM with deep learning techniques and does

Manuscript received September 7, 2023.
Manuscript revised December 4, 2023.
Manuscript publicized February 6, 2024.

[†]The authors are with the Department of Software Engineering, Jeonbuk National University, Korea Jeonju 54896, South Korea.
a) E-mail: funing230@jbnu.ac.kr
b) E-mail: duksan.ryu@jbnu.ac.kr (Corresponding author)
c) E-mail: stkim@jbnu.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2023EDP7183

not effectively address the change-point problem.

To address this issue, we introduced the MuSRGM, a novel model that integrates Genetic Algorithms (GA) and deep learning-based SRGMs. Our approach involves decomposing deep learning-based SRGMs into varied hidden layers, each defined by distinct activation functions. GA then dynamically combines these layers via a fitness function to generate the optimized MuSRGM.MuSRGM's innovation extends to its approach to failure data, treating it as a time series for comprehensive analysis. Embedded within MuSRGM, Long Short-Term Memory (LSTM) networks and an attention mechanism collaboratively work to unravel the nuances of failure data. LSTM is employed to discern the overall trends in the distribution of failure data, while the attention mechanism is adept at identifying anomalies within this time series that may signal change-points. Such an integration bolsters MuSRGM's ability to navigate the complexities inherent in software testing, particularly in contexts where change-points are present.

MuSRGM demonstrated superior prediction accuracy on three diverse datasets, as evidenced by a significant enhancement in mean squared error (MSE) performance, showing improvements of 71.51%, 41.39%, and 96.02% respectively, surpassing the baseline deep learning-based SRGMs. These findings highlight the effectiveness of MuSRGM in improving prediction performance. The consistent enhancement in MSE across diverse datasets underscores MuSRGM's robust generalization capability, affirming its efficacy in software reliability prediction. Unlike other research methods that match data using traditional SRGMs, MuSRGM starts from the data and customizes the model based on the unique characteristics of the failure dataset using GA. As a result, MuSRGM surpasses the performance of existing methods documented in the literature.

The main contributions of this paper are summarized as follows:

- The proposed method utilizes a genetic algorithm to dynamically combine the activation functions of deep learning-based SRGMs. This allows for the generation of MuSRGM, tailored to the specific features of the failure data.
- To further improve the accuracy of prediction, the MuSRGM incorporates both Long Short-Term Memory (LSTM) and Attention mechanisms. These techniques enable the SRGMs to effectively capture the irregular variation patterns of the failure data, leading to more accurate identification of change-point phenomena in the data.
- We separately examined the performance of combining the Attention and LSTM mechanisms with neural networks for analyzing failure datasets. The experimental results demonstrate that combining the Attention and LSTM mechanisms can further improve the predictive accuracy of neural networks for failure datasets.

The organization of the remaining sections in this paper is as follows. Section 2 offers a background and literature review for this study. In Sect. 3, we present the implementation details of the proposed method. Section 4 outlines the experimental setup and analyzes the results obtained from the experiments. The conclusion of this paper is presented in Sect. 5.

2. Background and Related Work

In this section, we aim to provide a comprehensive overview of the background knowledge necessary to understand the proposed approach for MuSRGM. The background to MuSRGM included the concepts of SRGMs, the change-point phenomenon, and Deep Learning-based SRGMs.

2.1 SRGMs

Software reliability growth models assume that the occurrence of software failures is a stochastic process, so traditional SRGMs are characterized as NHPP functions with specific parameters. These function parameters provide valuable information, such as the expected total number of failures in a project, the rate at which failures are detected, and so on. These functions can be used in future tests for future failure rates or the number of defects remaining in the code [17], [18]. One of the most well-known SRGMs is the Goel-Okumoto (GO) Model, which was introduced by Goel and Okumoto in 1979 and follows the equation:

$$m(t) = a(1 - e^{-bt}), a > 0, b > 0 \tag{1}$$

Where a is the total expected number of failures within the software project, b is the rate of failure discovery, and $m(t)$ is the expected cumulative number of failures discovered in the corresponding time.

2.2 Change-Point

Traditional SRGMs assume that FDR stays a continuous

Table 1 GO sub function

	GO Sub Function	GO Output
Input	t	t
Subfunction 1	sub1(t)	e^{-bt}
Subfunction 2	sub2(sub1(t))	$1-e^{-bt}$
Subfunction 2	sub3(sub2(sub1(t)))	$a(1-e^{-bt})$

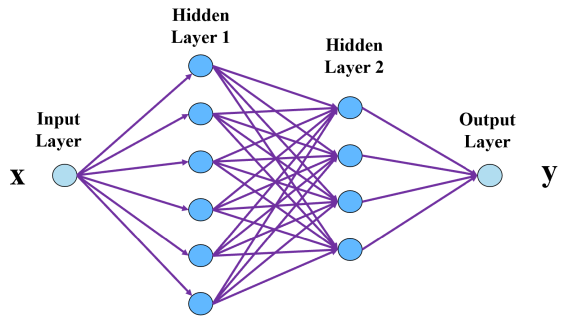


Fig. 2 Neural network structure

Table 2 Neural network layers

	Neural Network	Neural Network Output
`Input Layers`	x	x
`Hidden Layers 1`	activation1(x)	$w_{11}{}^0x+b_0$
`Hidden Layers 2`	activation2(activation1(x))	$w_{11}{}^1x(w_{11}{}^0x+b_0)+b_1$
`Output Layers`	activation3(activation2(activation1(x)))	$w_{11}{}^2(w_{11}{}^1x(w_{11}{}^0x+b_0)+b_1)+b_2$

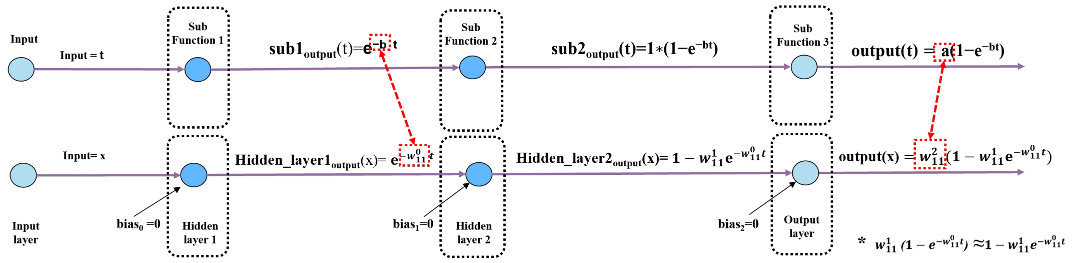

Input
Input = t
Sub Function 1
$\mathbf{sub1}_{\mathbf{output}}(t)=\mathbf{e}^{\boxed{-\mathbf{b}}\,t}$
Sub Function 2
$\mathbf{sub2}_{\mathbf{output}}\mathbf{(t)=1*(1-e^{-bt})}$
Sub Function 3
$\mathbf{output(t)} = \boxed{\mathbf{a}}\mathbf{(1-e^{-bt})}$
Input= x
$\mathbf{Hidden\_layer1}_{\mathbf{output}}(x)=\mathbf{e}^{\boxed{-w_{11}^0}\,t}$
$\mathbf{Hidden\_layer2}_{\mathbf{output}}\mathbf{(x)=1}-w_{11}^1\mathbf{e}^{-w_{11}^0 t}$
$\mathbf{output(x)} = \boxed{w_{11}^2}(1-w_{11}^1\mathbf{e}^{-w_{11}^0 t})$
Input layer
$\text{bias}_0=0$
Hidden layer 1
$\text{bias}_1=0$
Hidden layer 2
$\text{bias}_2=0$
Output layer
$* \; w_{11}^1\,(1-e^{-w_{11}^0 t}) \approx 1-w_{11}^1 e^{-w_{11}^0 t}$

Fig. 3 Neural network simulation

growth pattern throughout the testing phase, correlating with the count of remaining system failures. However, this assumption may not hold true in actual testing scenarios. The FDR can vary due to factors like changes in testing teams, methodologies, or equipment, leading to what are known as 'change-points'. These change-points, representing abrupt deviations in failure detection trends, demonstrate the nonlinear nature of software testing and challenge the efficacy of traditional SRGMs. This highlights the need for more adaptive and versatile models to accurately assess software reliability.

2.3 Deep Learning-Based SRGMs

In the pursuit of refining parameter estimation for SRGMs, a significant stride was taken by Wu and Huan [16]. They introduced an innovative paradigm through Deep Learning-based SRGMs and extensively elucidated the process of transforming conventional SRGMs into deep learning models. We take the Goel-Okumoto (GO) Model as an example to illustrate the method. The GO model is given by Eq. (1).

The equation above can be interpreted as a composite function with a variable t, which can be decomposed into three distinct sub-functions. This is shown in the following Table 1.

A deep learning network consists of an input layer, a hidden layer, and an output layer [19]. As shown in Fig. 2. Therefore, the deep learning network can be considered as a composite function mapped from the input x to the output y. The expression is shown as follows: y= *activation3*(*activation2*(*activation1*(x))) [20]. Where x is the input value of the neural network and y is the output value of the neural network [21]. $activation1()$, $activation2()$, and $activation3()$ are activation functions of the hidden layers of the neural network. Assuming that the bias of each hidden is b, the neural networks can be listed in the following Table 2.

It can be seen that the GO model has a similar structure to the Neural network, so the weights of the Neural network can be used to simulate the parameters a, and b in the GO model [22]. The specific method is shown in Fig. 3. With the weights $w_{11}{}^2$ corresponding to a and $w_{11}{}^1$ corresponding to b. In this way, the parameter values of the SRGMs can be determined by the Neural network.

2.4 Related Work

Various methods have been tried to predict software reliability, traditionally focusing on statistical and model-based approaches. Nevertheless, with the dynamic nature of software development, identifying and adapting to change-points has become crucial for accurate reliability predictions. Inoue [23] introduced a modeling framework that employs change-points to evaluate software reliability, noting significant shifts in hazard rates associated with these points. Zhao [13] developed an SRGM that integrates change-points with environmental factors from different testing phases, enhancing the model's reflection of actual testing environments. Samal et al. [24] further expanded this concept, considering the effects of incomplete debugging and change-points on SRGMs, and demonstrated impressive accuracy in fitting failure data.

In parallel with these developments, the advancement of AI technologies, especially deep learning, has led to a new wave of software reliability prediction methods. Lo [25] improved the accuracy of reliability predictions by directly feeding failure data into a Feed-Forward Neuro Network (FFNN). Cai et al. [26] divided failure data into time segments $X_1, X_2, X_3, \ldots, X_i, X_{i+1}$ and used $X_1, X_2, X_3, \ldots, X_i$ as input to a neural network to predict X_{i+1}. As RNNs matured, Fu et al. [27] and Gusmanov [28] both attempted to analyze failure data and predict software reliability using LSTM, achieving good results. Su [29], Wang [17], and Lakshmanan [19] have contributed to the combination of traditional SRGMs with neural networks. Wu and Huan [16] analyzed the characteristics of SRGMs and decomposed them into activation functions in neural networks, completing the transformation from traditional SRGMs to deep learning-based SRGMs.

After a thorough analysis of relevant studies, this paper proposes a method that integrates GA with deep learning-based SRGMs. Our approach fully exploits the properties of different SRGMs (concave and S-shaped types) and does not rigidly rely on fixed models to fit data, but instead generates models dynamically based on data characteristics. Consequently, it can fit the change-point phenomenon well.

3. Proposed Approach

In this section, we provide a detailed description of the MuSRGM. We begin with an overview of the research methodology, as illustrated in Fig. 4, followed by an explanation of each step, including data collection, data processing, Individual evaluation, and GA operation. The change-point phenomenon happens when there is a sudden change in the FDR, which violates the assumption of a continuous FDR throughout the testing phase. Therefore, a single S-shaped or Concave model cannot accurately fit the data distribution of the failure dataset. By combining the characteristics of S-shaped and Concave models, a better approximation of the failure distribution in real testing situations can be achieved. MuSRGM combines GA, deep learning-based SRGMs, LSTM, and Attention techniques to fully leverage the advantages of different technologies and enhance the accuracy of software reliability prediction. Firstly, the failure data is a time-series sequence that can be learned using LSTM to capture the inherent connections between the data. The role of the Attention mechanism is to identify anomalies in the time series, i.e., the change-point phenomenon. By combining GA with deep learning-based SRGMs, GA dynamically selects the best activation function combination based on the characteristics of the failure data, resulting in the optimal reliability assessment model.

Algorithm 1 shows the pseudo-code for MuSRGM. The input data consists of three parts: the failure dataset, the activation set of deep learning-based SRGMs, and the hyperparameters set required for training deep learning models. The activation function set and the set of hyperparameters are combined as the GA initialization population. The fitness function is used to calculate the fitness value of each individual in the population, and the parameters of the neural network with the lowest fitness value are output as the result when the stopping condition of the GA operation is met.

3.1 Data Collection

Before using the GA algorithm, we need to gather the data. This data consists of two types: (1) the failure dataset used for training the deep learning model, and (2) the set of parameters for the GA algorithm. The failure dataset consists of two features: the time of failure detection and the cumulative number of failures, which are illustrated in Fig. 5. The time of failure detection can be represented by calendar time, system running time, or the number of tests.

Recording the calendar time of failures during testing can be impractical and expensive. Additionally, testing is

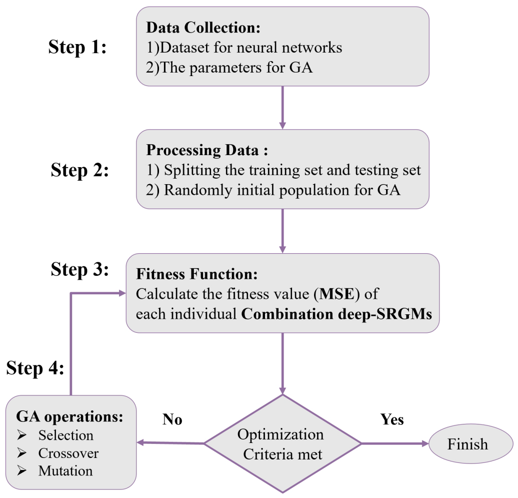

Step 1:
Data Collection:
1)Dataset for neural networks
2)The parameters for GA
Step 2:
Processing Data :
1) Splitting the training set and testing set
2) Randomly initial population for GA
Step 3:
Fitness Function:
Calculate the fitness value (MSE) of each individual Combination deep-SRGMs
Step 4:
GA operations:
Selection
Crossover
Mutation
No
Optimization Criteria met
Yes
Finish

Fig. 4 Overview of proposed method

Algorithm 1 MuSRGM Algorithm

```
Input:
    /* Failure dataset                                        */
  $Dataset_{failure}$
    /* The set of activation functions                        */
  $Population_{SRGMs\_activation}$
    /* Deep learning training hyper_parameter set             */
  $Population_{Hyper\_parameters}$
Output:
    /* MuSRGM activation functions and hyper_parameters */
  $Best\_individual$
Initialize:
  $TrainingDataset, TestingDataset = Splitting(Dataset_{failure})$;
  $iter \leftarrow 0$ ;
while Optimization_Criteria=False do
    /* The GA begins by initializing the population    */
    foreach $individual \in$
      $Population_{activation} \cup Population_{Hyper\_parameters}$ do
        Random initialize $individual$=
          $\{SRGMs\_activation\_1\_iter, SRGMs\_activation\_2\_iter,$
          $Batchsize, LossFunction, \cdots\}$
    /* Build deep learning model                          */
    $input = input\_layer(Dataset.shape)$
      /* Add LSTM layer                                   */
    $LSTM\_layer = LSTM(input)$
      /* Add Attention layer                              */
    $Attention\_layer = Attention(LSTM\_layer)$
      /* Add hidden_layer1 to the current population   */
    $hidden\_layer1 = individual\_activation\_1(Attention\_layer)$
      /* Add hidden_layer2 to the current population   */
    $hidden\_layer2 = individual\_activation\_2(hidden\_layer1)$
      /* Add output_layer to the current population    */
    $output = individual\_activation\_3(hidden\_layer2)$
    $model = Neural\_Network(input, output)$
    $model.compile(optimizer, loss\_function)$
    $model.fit(TrainingDataset)$
    $predict\_value = model.predict(TestingDataset)$
    $fitness = Fitness\_Function(predict\_value, true\_value)$
    $population_{iter+1} \leftarrow Generating\_Population$
    $(Selection, Crossover, Mutation)$ $iter \leftarrow iter + 1$
$Return(The\_best\_individual\_from\_the\_populations)$
```

often conducted asynchronously, and the probability of detecting similar types of failures can vary from test to test. Therefore, system running time is not a good metric. In-

T	1	2	3	4	5	6	7	8	…	204	205	206	207	208
CFC	318	708	902	1860	2403	3131	3937	4943	…	8776	8776	8777	8777	8780

T: Time
CFC: Cumulative Failure Count

Fig. 5 Failure data for eclipse's JDT project

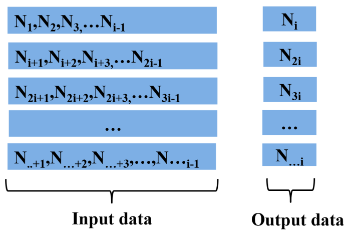
Table 3 The set of activation functions

Traditional SRGMs		Hidden layer 1		Hidden layer 2		Output layer	
Name	Formula	Activation	Bias	Activation	Bias	Activation	Bias
LGC	$m(t) = \frac{a}{1+ke^{-bt}}, a, b, k > 0$ a: The total number of failures to be eventually detected b, k: Parameters predicting future trends, derived from historical data	math.exp(-x)	0	math.pow(1+x,-1)	0	x	0
GO	$m(t) = a(1 - e^{-bt}) \quad a > 0, b > 0$ a: The expected total number of failures to be detected b: Fault detection rate	math.exp(-x)	0	math.subtract(1,x)	0	x	0
DSS	$m(t) = a\left(1 - (1 + ct)e^{-bt}\right) \quad a, b, c > 0$ a: The expected total number of failures to be detected b: The fault detection rate c: The rate of developer progress	math.exp(-x)	0	1-x	1	x	0
Musa	$m(t) = a\ln(1 + bt) \quad a, b > 0$ a: The expected total number of failures to be detected b: Parameter for converting calendar time to execution time	x	1	math.log(1+x)	0	x	0
YEX	$m(t) = a\left(1 - e^{-r(1-e^{-bt})}\right) \quad a, b, r > 0$ a: The expected total number of failures to be detected b: The fault detection rate r: Total effort to be expended	math.subtract(1,math.exp(x))	0	math.subtract(1,math.exp(x))	0	x	0

stead, a fixed period of system operation is the most suitable metric for counting the number of failures. The cumulative number of failures refers to the count of unique failures identified in the code during a specific system run period.

The parameter set of our GA-based method comprises two types of data: activation functions from deep learning-based SRGMs, and hyperparameters for model training. Following Wu and Huan [16], We transform traditional SRGMs into deep learning models with three hidden layers, each incorporating activation functions as part of the parameter set. Given the classification of traditional SRGMs into Concave and S-shaped models according to the FDR, our three-layer deep learning-based SRGMs are designed to embody these patterns. We adopted various models, such as the Logarithmic Growth Curve (LGC), Goel-Okumoto (GO), Delayed S-shaped Curve (DSS), Musa–Okumoto Logarithmic Poisson (Musa), and Yamada Exponential (YEX), to represent the spectrum of SRGMs, each with significant applications and research in software reliability. Specifically, the LGC and GO models, characteristic of their concave nature, are optimal for the early defect discovery stage when the discovery rate declines as testing advances. Conversely, DSS, Musa, and YEX, as S-shaped models, effectively capture the incremental complexity at different testing phases. Table 3 details these models' activation functions.

MuSRGM combines deep learning-based SRGMs with LSTM networks and attention mechanisms. Key hyperparameters, such as learning rate, training epochs, batch size, and time steps, are optimized using GA for maximum efficacy. The objective of MuSRGM is to blend the features of both Concave and S-shaped models, effectively fitting the change-point phenomenon by systematically combining model elements and hyperparameters.



i : Variable time steps.
N : Total cumulative number of failures found in a fixed system operating cycle.

Fig. 6 Input and output data shape

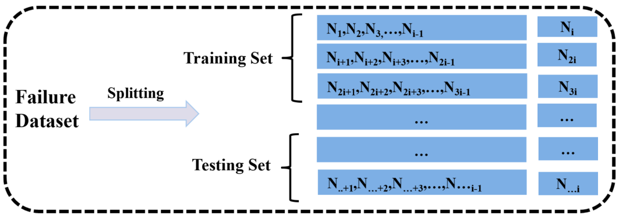
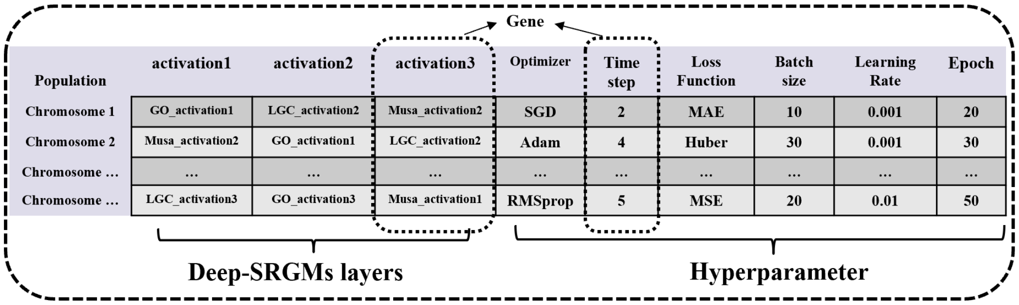


Fig. 7 Splitting dataset

3.2 Processing Data

Our goal is to predict the number of failures that will occur in future system operating cycles based on the cumulative number of failures provided by the failure dataset. To achieve this, we use the LSTM model, which treats the failure data as a time series. Each input is a vector of failure data, and the output is a single value representing the number of failures at the next time. Figure 6 shows the data shape of the input and output data. To train and test the deep learning model,



Population	activation1	activation2	activation3	Optimizer	Time step	Loss Function	Batch size	Learning Rate	Epoch
Chromosome 1	GO_activation1	LGC_activation2	Musa_activation2	SGD	2	MAE	10	0.001	20
Chromosome 2	Musa_activation2	GO_activation1	LGC_activation2	Adam	4	Huber	30	0.001	30
Chromosome …	…	…	…	…	…	…	…	…	…
Chromosome …	LGC_activation3	GO_activation3	Musa_activation1	RMSprop	5	MSE	20	0.01	50

Fig. 8 Initial population

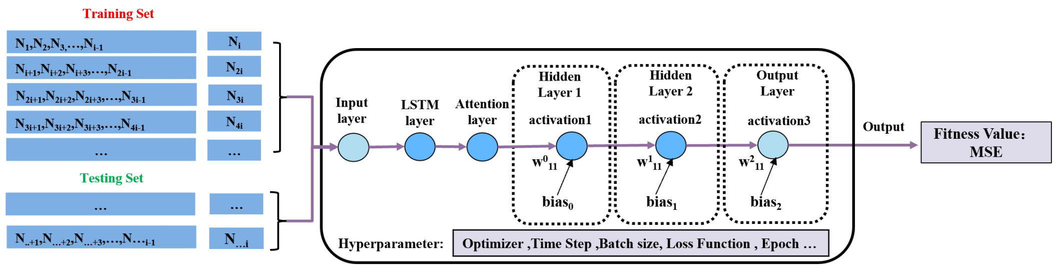
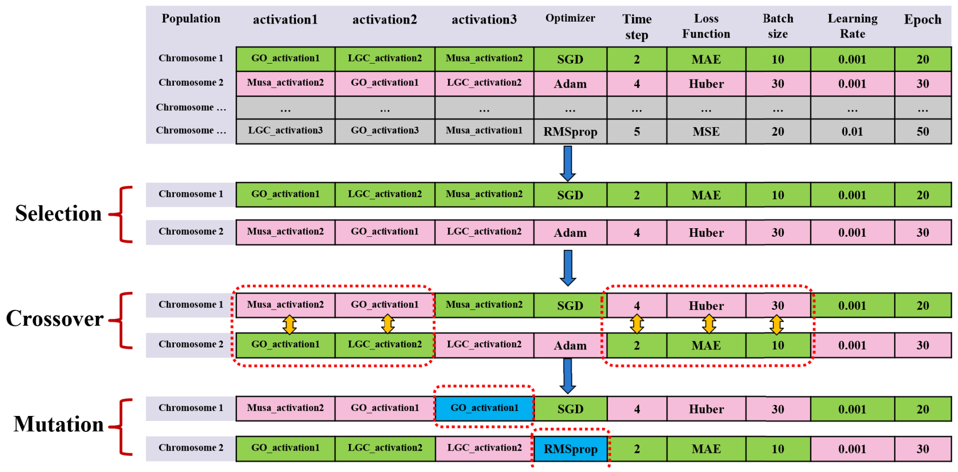

Training Set
$N_1,N_2,N_3,\ldots,N_{i-1}$
N_i
$N_{i+1},N_{i+2},N_{i+3},\ldots,N_{2i-1}$
N_{2i}
$N_{2i+1},N_{2i+2},N_{2i+3},\ldots,N_{3i-1}$
N_{3i}
$N_{3i+1},N_{3i+2},N_{3i+3},\ldots,N_{4i-1}$
N_{4i}
…
Testing Set
$N_{\ldots+1},N_{\ldots+2},N_{\ldots+3},\ldots,N_{\ldots i-1}$
$N_{\ldots i}$
Input layer
LSTM layer
Attention layer
Hidden Layer 1
activation1
w^0_{11}
$bias_0$
Hidden Layer 2
activation2
w^1_{11}
$bias_1$
Output Layer
activation3
w^2_{11}
$bias_2$
Output
Fitness Value: MSE
Hyperparameter: Optimizer ,Time Step ,Batch size, Loss Function , Epoch …

Fig. 9 Fitness function



Population	activation1	activation2	activation3	Optimizer	Time step	Loss Function	Batch size	Learning Rate	Epoch
Chromosome 1	GO_activation1	LGC_activation2	Musa_activation2	SGD	2	MAE	10	0.001	20
Chromosome 2	Musa_activation2	GO_activation1	LGC_activation2	Adam	4	Huber	30	0.001	30
Chromosome …	…	…	…	…	…	…	…	…	…
Chromosome …	LGC_activation3	GO_activation3	Musa_activation1	RMSprop	5	MSE	20	0.01	50

Selection

Chromosome 1	GO_activation1	LGC_activation2	Musa_activation2	SGD	2	MAE	10	0.001	20
Chromosome 2	Musa_activation2	GO_activation1	LGC_activation2	Adam	4	Huber	30	0.001	30

Crossover

Chromosome 1	Musa_activation2	GO_activation1	Musa_activation2	SGD	4	Huber	30	0.001	20
Chromosome 2	GO_activation1	LGC_activation2	LGC_activation2	Adam	2	MAE	10	0.001	30

Mutation

Chromosome 1	Musa_activation2	GO_activation1	GO_activation1	SGD	4	Huber	30	0.001	20
Chromosome 2	GO_activation1	LGC_activation2	LGC_activation2	RMSprop	2	MAE	10	0.001	30

Fig. 10 GA operations

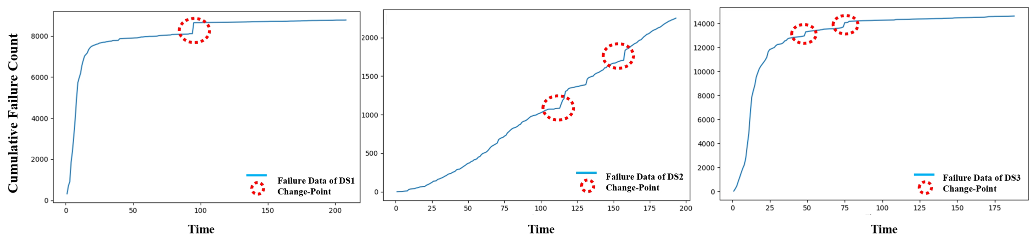
we divide the failure dataset into a training set (90%) and a testing set (10%), as illustrated in Fig. 7.

When initiating the GA algorithm, population initialization is necessary [30]. Figure 8 illustrates the exact execution process. Each chromosome (individual solution) consists of two parts: the set of activation functions of deep learning-based SRGMs and the set of hyperparameters for the model [31]. We specify the corresponding search space for each parameter. Random values are assigned to each parameter in the search space during the initial run of the GA algorithm.

3.3 Individual Evaluating

After each iteration of the genetic algorithm, the next generation of the population is generated. The offspring need to be evaluated using a fitness function [32], [33]. The fitness function assesses the proximity of an individual solution to the ideal solution [34], and we use Mean Squared Error (MSE) as the criterion for the fitness function. A lower MSE indicates a higher fitness of the solution. MuSRGM is also integrated into the fitness function, as shown in Fig. 9. It includes a dynamic combination SRGM consisting of an LSTM layer, an Attention layer, and three hidden layers composed of activation functions extracted from deep learning-based SRGMs.

The combination of the LSTM and Attention mechanism is designed to capture the change-point phenomenon that occurs during the testing process. In the GA algorithm, each iteration of the fitness function builds an SRGM model



(a) DS1 (b) DS2 (c) DS3

Fig. 11 Cumulative failure curves for DS1, DS2, DS3

using the current solution (a single chromosome). The training dataset is passed to the SRGM model for training, and after the model is trained, the test dataset is used to calculate the MSE value (fitness value). This process is repeated until the GA termination condition is met. The model with the smallest fitness value (MSE value) is the best model generated by the GA algorithm.

3.4 GA Related Operating

When the termination condition of the GA algorithm is not met, it undergoes the selection, Crossover, or Mutation operation to generate the next generation and calls the fitness function again for evaluation. This process is repeated until the termination condition is met. The execution process of selection, Crossover, or Mutation is illustrated in Fig. 10. Once the termination condition is satisfied, the model with the lowest fitness value is considered the best model, and the associated parameters of this model are the final output.

4. Evaluation

To validate the proposed model, the following research questions were developed.

RQ1: How does the proposed method compare to other state-of-the-art methods in terms of fitting the change-point phenomenon?

RQ2: How does the proposed method perform on different failure data sets in terms of Generalizability?

RQ3: Does the addition of the attention mechanism effectively improve the prediction performance of the SRGMs?

RQ1 aims to verify the effectiveness of our proposed method in fitting the change-point phenomenon compared to other state-of-the-art methods. RQ2 investigates the generalizability of the proposed method for different failure datasets. RQ3 examines whether the addition of the attention mechanism can effectively enhance the predictive performance of the model.

4.1 Experimental Setup

4.1.1 Datasets

We validate the proposed model using three datasets: Eclipse's Java Development Tools (JDT) project, the networking component of the Linux Kernel, and Eclipse's platform project. Detailed descriptions of the datasets are presented in Table 4.

Table 4 Failure dataset

label	Dataset	Duration	NO.of Failures	References
DS1	Eclipse's Java Development Tools (JDT) Project	6334 days	8780	[35]
DS2	The Networking Component of The Linux Kernel	5993 days	2251	[36]
DS3	Eclipse's Platform Project	5715 days	14608	[35]

Figure 11 displays the cumulative failure curve graphs for three datasets. It can be observed that FDR does not follow a constant curve. At a specific point during testing (indicated by the dashed circle), the FDR suddenly changes. Figure 11 (a) shows a noticeable change in FDR during the middle of the test. Additionally, Fig. 11 (b) and (c) indicate that two distinct changes in FDR occur during the testing process. This phenomenon is known as a change-point, which frequently occurs during the practical testing phase.

During the software testing phase, change-points may arise from intentional or unintentional factors. However, the large project datasets we rely on do not provide specific classification information on change-points. Nevertheless, the flexible design of the MuSRGM model enables it to analyze and capture all types of change-points. This model leverages the combination of comprehensive analysis of historical failure data with the strengths of deep learning and genetic algorithms to identify significant changes during the testing process, regardless of whether these changes are planned or random.

4.1.2 Evaluation Metrics

We compare the performance of the proposed model with other models using several evaluation criteria.

1. Mean-Square Error (MSE): A measure of the difference between the predicted and actual values, calculated as the average squared difference between them. It is a non-negative value, and a lower MSE indicates a better fit between the predicted and actual values. The MSE is calculated as follows:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

2. Mean Absolute Percentage Error (MAPE): A metric that uses statistics to evaluate prediction accuracy. It is calculated as follows:

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{Y_i - \hat{Y}_i}{Y_i}\right|$$

3. Symmetric Mean Absolute Percentage Error (SMAPE): A metric that uses percentage error to evaluate the expected accuracy, which is calculated as follows:

$$SMAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\left|\hat{Y}_i - Y_i\right|}{(|Y_i| + |\hat{Y}_i|)/2}\right|$$

4. Mean Percentage Error (MPE): A measure that uses statistics to evaluate the mean of the percentage error between the predicted and actual values:

$$MPE = \frac{100\%}{n}\sum_{i=1}^{n}\frac{Y_i - \hat{Y}_i}{Y_i}$$

In the formula above,$\hat{Y}_i$ represents the number of failures predicted using the model at the time i, Y_i represents the true number of failures at the time i, and n represents the total amount of data.

5. Mean Time Between Failures (MTBF): A measure that calculates the average time elapsed between two consecutive failures in a system, which is calculated as follows:

$$MTBF = \frac{T_{\text{total}}}{N}$$

T_{total} represents the total running time during the observation period, and N is the total number of failures observed in the system.

4.1.3 Baseline Methods

In comparison with related work, we establish baseline methods using the research methods mentioned and use these baselines to evaluate the superiority of our model performance. We establish a total of four baseline methods.

1. Neural network
Traditional SRGMs rely on formulas with parameters for interpretation, but due to the limitations of these parameters, none of them can achieve accurate predictions on all failure data. In contrast, neural networks use weights to adjust the gap between input data and labeled data without the same limitations on parameters, leading to higher accuracy.

2. LSTM
LSTM is a type of RNN network that can predict future outcomes based on previous knowledge, and it can capture the intrinsic relationship between data, which results in higher prediction accuracy.

3. Deep learning-based SRGMs
Deep learning-based SRGMs combine the characteristics of traditional SRGMs and neural networks. They simulate the parameters of SRGMs using the weights of neural networks, and adjust the influence of each SRGM on the final output through error backward propagation, making them more adaptable to different failure data sets.

4. GA-based SRGMs
Before the application of AI techniques in SRGMs, GA was a popular approach for parameter optimization. In this study, GA is used in combination with five SRGMs, and the resulting prediction results are used as one of the baseline methods for comparison.

5. SRGM with change-point
SRGM with change-point comprehensively analyzes the dynamics of software reliability by incorporating FDR before and after the change-points. This integration provides a holistic perspective on how software reliability evolves, taking into account variations in FDR during different phases of testing.

4.2 Experimental Result

RQ1: How does the proposed method compare to other state-of-the-art methods in terms of fitting the change-point phenomenon?

The purpose of this investigation is to evaluate the effectiveness of our proposed method. We compared the predictive performance of GA-based models, neural networks, LSTM networks, and MuSRGM on three real-world datasets. The prediction results are presented in Tables 5-7. Upon reviewing the results, it is evident that the traditional SRGMs based on GA exhibit poor performance for the three datasets, with an MSE that is one order of magnitude higher than all deep learning-based methods. However, there are also significant differences in the MSE of each deep learning-based method. The LSTM-based method demonstrates the poorest performance, as evidenced by the highest MSE score among all models for dataset 1 (3372.14, as shown in Table 5). The MSE differences among the various methods are not significant for dataset 2, and the prediction differences are relatively small. For dataset 3, the deep learning-based SRGMs exhibit the worst performance with the highest MSE score of 7078.87 (as shown in Table 7). After observing the three datasets, MuSRGM demonstrates superior performance compared to all baseline methods, validating the effectiveness of combining SRGMs dynamically based on the characteristics of failure data in improving prediction accuracy. In summary, these findings confirm that our proposed method outperforms all other methods in the literature. The MuSRGM model yields identical values for SMAPE and MPE across all three datasets, as shown in Tables 5, 6, and 7 (0.000589, 0.001192, and 0.000388, respectively). This is due to the small range of data values and the relatively even distribution of prediction errors compared to true values. This further confirms that the predictive performance of MuSRGM is better than other methods.

Furthermore, prediction accuracy is gauged by the proximity of predicted MTBF to actual test data MTBF. An analysis of the data presented in Table 8 reveals a significant edge of Deep learning-based SRGMs over those GA and those incorporating change-points. This discovery not only highlights the exceptional predictive capabilities of neural networks but also emphasizes their effectiveness and adapt-

Table 5 Experimental result for DS1

Model	MSE	MAPE	SMAPE	MPE
GO model(GA)	109,724.93	3.78	3.85	-3.78
DSS model(GA)	154,508.36	4.48	4.58	-4.48
LGC model(GA)	182,232.19	4.87	4.99	-4.87
Musa model(GA)	299,190.58	6.23	6.04	6.23
YEX Model(GA)	55,700.05	2.69	2.73	-2.69
SRGM with change-points	48,262.90	2.60	2.10	-2.19
Neural Network	198.62	0.16	0.16	0.16
LSTM	3372.14	0.66	0.66	0.66
Deep learning-based SRGMs	479.28	0.23	0.23	0.23
MuSRGM	136.51	0.001179	0.000589	0.000589

Table 6 Experimental result for DS2

Model	MSE	MAPE	SMAPE	MPE
GO model(GA)	32,602.27	8.41	8.77	-8.41
DSS model(GA)	2150.97	2.15	2.17	-2.15
LGC model(GA)	34,942.97	8.55	8.94	-8.55
Musa model(GA)	32,709.86	8.42	8.79	-8.42
YEX Model(GA)	32,960.93	8.45	8.82	-8.45
SRGM with change-points	31,153.51	8.32	8.42	-8.32
Neural Network	210.77	3.08	3.06	-0.66
LSTM	109.75	3.09	3.08	-0.47
Deep learning-based SRGMs	107.84	0.42	0.42	-0.06
MuSRGM	63.20	0.002377	0.001192	0.001192

Table 7 Experimental result for DS3

Model	MSE	MAPE	SMAPE	MPE
GO model(GA)	48,065.06	1.51	1.51	-1.51
DSS model(GA)	243,702.17	3.38	3.44	-3.38
LGC model(GA)	373,635.96	4.19	4.28	-4.19
Musa model(GA)	1,942,125.23	9.54	9.11	9.54
YEX Model(GA)	2,120,344.85	3.16	3.21	-3.17
SRGM with change-points	1,107,869.30	8.12	8.42	8.32
Neural Network	2817.82	0.36	0.36	-0.36
LSTM	3330.25	0.39	0.39	-0.39
Deep learning-based SRGMs	7078.87	0.56	0.56	0.56
MuSRGM	281.68	0.000777	0.000388	0.000388

Table 8 MTBF comparison

Model	MSE	MAPE	SMAPE	MPE
GO model(GA)	48,065.06	1.51	1.51	-1.51
DSS model(GA)	243,702.17	3.38	3.44	-3.38
LGC model(GA)	373,635.96	4.19	4.28	-4.19
Musa model(GA)	1,942,125.23	9.54	9.11	9.54
YEX Model(GA)	2,120,344.85	3.16	3.21	-3.17
SRGM with change-points	1,107,869.30	8.12	8.42	8.32
Neural Network	2817.82	0.36	0.36	-0.36
LSTM	3330.25	0.39	0.39	-0.39
Deep learning-based SRGMs	7078.87	0.56	0.56	0.56
MuSRGM	281.68	0.000777	0.000388	0.000388

ability in handling complex data patterns. Importantly, MuSRGM's MTBF predictions closely match the actual values of 0.8234, 0.0706, and 0.2500 for DS1, DS2, and DS3, respectively, with predicted values of 1.2271, 0.0787, and 0.4104. This demonstrates MuSRGM's accuracy in fitting the change-point phenomenon and highlights its predictive precision.

RQ2: How does the proposed method perform on different failure data sets in terms of Generalizability?

The purpose of this research question is to assess the generalizability of our proposed method across different datasets. The individual prediction results in Tables 5-7 indicate significant variations in model performance across different datasets. We found that GA-based SRGMs generally exhibited high MSE values and poor performance across all three datasets. In contrast, deep learning-based methods demonstrated significant differences in predicting models for different datasets. For instance, while the LSTM model demonstrated a low MSE score on dataset 2 (109.75, as shown in Table 6), it performed poorly on datasets 1 and 3. Conversely, the deep learning-based SRGMs exhibited strong performance on datasets 1 and 2 but fared poorly on dataset 3, as evidenced by its high MSE score of 7078.87 (as shown in Table 7). Therefore, no single model could perform well on all datasets. However, MuSRGM employs natural selection to search the parameter space and uses a fitness function to determine the best solution. Consequently, our proposed method performed well on all three datasets: DS1, DS2, and DS3. We conclude that our proposed method is more applicable than existing methods in the literature.

Table 9 Comparison of the effects of attention mechanism on DS1

Model	MSE	MAPE	SMAPE	MPE
Neural Network	198.62	0.16	0.16	0.16
LSTM	3372.14	0.66	0.66	0.66
Neural Network With Attention	1154.08	0.39	0.39	0.38
LSTM With Attention	227.59	0.17	0.17	0.17

Table 10 Comparison of the effects of attention mechanism on DS2

Model	MSE	MAPE	SMAPE	MPE
Neural Network	210.77	3.07	3.06	-0.66
LSTM	109.75	3.09	3.08	-0.47
Neural Network With Attention	529.24	3.18	3.15	-1.06
LSTM With Attention	23.70	2.88	2.87	-0.17

Table 11 Comparison of the effects of attention mechanism on DS3

Model	MSE	MAPE	SMAPE	MPE
Neural Network	2817.82	0.36	0.36	-0.36
LSTM	3330.25	0.39	0.39	-0.39
Neural Network With Attention	7477.00	0.59	0.59	-0.59
LSTM With Attention	1904.88	0.29	0.29	0.29

RQ3: Does the addition of the attention mechanism effectively improve the prediction performance of the models?

The purpose of this research question is to evaluate the effectiveness of adding the Attention mechanism to the software reliability prediction model. We apply the Neural network and LSTM without the Attention mechanism, as well as the Neural network and LSTM with the Attention mechanism, to three failure datasets and present the prediction results in Tables 9-11. The results for the Neural network and LSTM without the Attention mechanism show that the Neural network performs better than the LSTM. However, the addition of the Attention mechanism has a varying impact on the prediction performance of the models. Specifically, the combination of the Attention mechanism and Neural network yields worse results than using the Neural network alone. On the other hand, combining the Attention mechanism with the LSTM greatly improves its prediction performance. Based on the results in Table 9, the prediction performance of the LSTM with attention model is slightly lower than that of the neural network for DS1 (see 227.59). However, for DS2 and DS3, the prediction performance of the LSTM with attention model surpasses that of the neural network. These findings suggest that the combination of LSTM and attention can effectively capture the change-point phenomenon in time series data, leading to an improvement in the model's prediction performance.

4.3 Threats to Validity

This section addresses the limitations of our proposed method. As it is widely acknowledged, obtaining a failure dataset is the most challenging aspect of software reliability research. We endeavored to test the model's performance using several datasets, but as each dataset was sourced from different software development projects, the test results were either extremely positive or negative. To mitigate this, we evaluated the proposed model using the same dataset as the one proposed by Wu and Huan [16]. We hope that relevant institutions or companies can provide more accurate failure datasets, enabling us to make more precise predictions about software reliability.

Additionally, to validate the effectiveness of our model, we established 9 baselines for performance comparison. While implementing each baseline based on relevant research papers, there may be some discrepancies compared to the original methods proposed in those papers.

5. Conclusion

Controlling software development costs by shortening testing cycles while ensuring software quality is a major challenge facing today's software industry. SRGMs have gained attention for predicting the number of failures that may occur in future system operations, helping estimate the optimal time for system release. However, traditional SRGMs, which assume a continuous FDR throughout testing phases, are challenged by the occurrence of change-points; these events contradict these assumptions and compromise the accuracy of predictions as well as the reliability assessment of the software.

In this paper, we propose a novel approach for software reliability prediction by combining Deep learning-based SRGMs with LSTM and Attention mechanisms to capture the change-point phenomenon in failure data sets. The Activation function is extracted from the Deep learning-based SRGMs and used to form a new set of mutated SRGMs.These mutated SRGMs are then evaluated using GA to find the optimal SRGM model with the lowest MSE value. The new SRGMs take advantage of both Concave and S-shaped types, resulting in higher prediction accuracy and greater applicability.

We validated the proposed method with three different failure datasets and demonstrated that our model's prediction accuracy is better than that of the state-of-the-art research method. Furthermore, our proposed method achieved excellent prediction results for all three datasets, indicating its superior applicability compared to the state-of-the-art research method. This will help accurately evaluate software reliability, thereby shortening the testing cycle and saving software development costs.

Currently, the training data used in the proposed model only has a single feature, which is the cumulative number of failures. In order to further improve the predictive performance of the model, we will look for more failure data features to enrich the training dataset. For example, the type of failure, the severity level of the failure, and so on. Combining enriched data features and the proposed model will further improve the accuracy of the predictions.

Acknowledgments

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (RS-2023-00259099) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). Additionally, this research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2022R1I1A3069233).

References

[1] M.R. Lyu, "Software reliability engineering: A roadmap," Future of Software Engineering (FOSE '07), pp.153–170, 2007.
[2] V. Almering, M. van Genuchten, G. Cloudt, and P.J.M. Sonnemans, "Using software reliability growth models in practice," IEEE Softw., vol.24, no.6, pp.82–88, 2007.
[3] M.R.-T. Lyu, "Software reliability theory," Encyclopedia of Software engineering, vol.2, pp.1611–1630, 2002.
[4] E. Zio, "Reliability engineering: Old problems and new challenges," Reliability engineering & system safety, vol.94, no.2, pp.125–141, 2009.
[5] J. Zhang, Y. Lu, S. Yang, and C. Xu, "Nhpp-based software reliability model considering testing effort and multivariate fault detection rate," Journal of Systems Engineering and Electronics, vol.27, no.1, pp.260–270, 2016.
[6] A. Wood, "Software reliability growth models," Tandem Technical Report, vol.96, no.130056, p.900, 1996.
[7] N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Using neural networks in reliability prediction," IEEE Softw., vol.9, no.4, pp.53–59, 1992.
[8] 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.2019, no.1, pp.43–58, 2019.
[9] A.R. Pai, G. Joshi, and S. Rane, "Quality and reliability studies in software defect management: a literature review," International Journal of Quality & Reliability Management, 2021.
[10] C. Smidts, "Research in software reliability engineering," RAMS '06. Annual Reliability and Maintainability Symposium, 2006., pp.228–233, 2006.
[11] K.K. Raghuvanshi, A. Agarwal, K. Jain, and V.B. Singh, "A time-variant fault detection software reliability model," SN Applied Sciences, vol.3, no.1, pp.1–10, 2021.
[12] C.-Y. Huang and M.R. Lyu, "Estimation and analysis of some generalized multiple change-point software reliability models," IEEE Trans. Rel., vol.60, no.2, pp.498–514, 2011.
[13] J. Zhao, H.-W. Liu, G. Cui, and X.-Z. Yang, "Software reliability growth model with change-point and environmental function," Journal of Systems and Software, vol.79, no.11, pp.1578–1587, 2006.

[14] V. Dhaka and N. Nijhawan, "Effect of change in environment on reliability growth modeling integrating fault reduction factor and change point: a general approach," Annals of Operations Research, pp.1–35, 2022.

[15] M.K. Bhuyan, D.P. Mohapatra, and S. Sethi, "A survey of computational intelligence approaches for software reliability prediction," ACM SIGSOFT Software Engineering Notes, vol.39, no.2, pp.1–10, 2014.

[16] C.-Y. Wu and C.-Y. Huang, "A study of incorporation of deep learning into software reliability modeling and assessment," IEEE Trans. Rel., vol.70, no.4, pp.1621–1640, 2021.

[17] G. Wang and W. Li, "Research of software reliability combination model based on neural net," 2010 Second World Congress on Software Engineering, pp.253–256, 2010.

[18] J. Zheng, "Predicting software reliability with neural network ensembles," Expert systems with applications, vol.36, no.2, pp.2116–2122, 2009.

[19] I. Lakshmanan and S. Ramasamy, "An artificial neural-network approach to software reliability growth modeling," Procedia Computer Science, vol.57, pp.695–702, 2015.

[20] Y. Singh and P. Kumar, "Application of feed-forward neural networks for software reliability prediction," ACM SIGSOFT Software Engineering Notes, vol.35, no.5, pp.1–6, 2010.

[21] G. Wang and W. Li, "Research of software reliability combination model based on neural net," 2010 Second World Congress on Software Engineering, pp.253–256, IEEE, 2010.

[22] T. Yaghoobi, "Parameter optimization of software reliability models using improved differential evolution algorithm," Mathematics and Computers in Simulation, vol.177, pp.46–62, 2020.

[23] S. Inoue and S. Yamada, "Optimal software release policy with change-point," 2008 IEEE International Conference on Industrial Engineering and Engineering Management, pp.531–535, 2008.

[24] U. Samal, S. Kushwaha, and A. Kumar, "A testing-effort based srgm incorporating imperfect debugging and change point," Reliability: Theory & Applications, vol.18, no.1, pp.86–93, 2023.

[25] J.-H. Lo, "The implementation of artificial neural networks applying to software reliability modeling," Proc. 21st Annual International Conference on Chinese Control and Decision Conference, CCDC'09, pp.4385–4390, 2009.

[26] K.-Y. Cai, L. Cai, W.-D. Wang, Z.-Y. Yu, and D. Zhang, "On the neural network approach in software reliability modeling," Journal of Systems and Software, vol.58, no.1, pp.47–62, 2001.

[27] F. Yangzhen, Z. Hong, Z. Chenchen, and F. Chao, "A software reliability prediction model: Using improved long short term memory network," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp.614–615, IEEE, 2017.

[28] K. Gusmanov, "On the adoption of neural networks in modeling software reliability," Proc. 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.962–964, 2018.

[29] Y.-S. Su and C.-Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," Journal of Systems and Software, vol.80, no.4, pp.606–615, 2007.

[30] M. Saraswat, "Cost optimization of srgm using genetic algorithm," International Journal of Computer Applications, vol.144, no.5, pp.13–20, 2016.

[31] A. Tickoo, P.K. Kapur, S.K. Khatri, and A.K. Verma, "Optimal release time determination for multi upgradation srgm with faults of different severity using genetic algorithm," 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), pp.1–6, IEEE, 2015.

[32] J. Yang, M. Zhao, and W. Hu, "Web software reliability modeling with random impulsive shocks," Journal of Systems Engineering and Electronics, vol.25, no.2, pp.349–356, 2014.

[33] 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, p.2945, 2021.

[34] D.G. KrishnaMohan, B. Sowmya, K. Mohanvamsi, and K. Sandeep, "An effective software reliability estimation with real-valued genetic algorithm," International Journal of Engineering & Technology, vol.7, p.359, 2018.

[35] "Eclipse bugzilla," https://bugs.eclipse.org/bugs/, accessed March 12, 2022.

[36] "Linux kernel bugzilla," https://bugzilla.kernel.org/, accessed March 12, 2022.

Ning Fu earned his Bachelor's degree in Software Engineering from Dalian Jiaotong University, China, in 2004. He furthered his education by completing a Master's degree in Software Engineering at Nanjing University of Science and Technology, China, in 2016. Currently, he is dedicated to pursuing a Ph.D. at Chonbuk National University in South Korea. Between May 2009 and December 2020, Ning Fu held the position of Senior Lecturer at Henan Agricultural Vocational College, China.

Duksan Ryu earned a bachelor's degree in computer science from Hanyang University in 1999 and a Master's dual degree in software engineering from KAIST and Carnegie Mellon University in 2012. He received his Ph.D. degree from the School of Computing, KAIST in 2016. His research areas include software analytics based on AI, software defect prediction, mining software repositories, and software reliability engineering. He is currently an associate professor in the software engineering department at Jeonbuk National University.

Suntae Kim is a Full Professor in the Department of Software Engineering at Chonbuk National University. He earned his B.S. degree in computer science and engineering from Chung-Ang University in 2003, and both his M.S. and Ph.D. degrees in computer science and engineering from Sogang University in 2007 and 2010, respectively. His research focuses on software architecture, artificial intelligence, blockchain, and financial technology.