International Journal of Mathematical, Engineering and Management Sciences
Vol. 7, No. 3, 325-340, 2022
https://doi.org/10.33889/IJMEMS.2022.7.3.022

Software Reliability Prediction through Encoder-Decoder Recurrent Neural Networks

Chen Li
Department of Bioscience and Bioinformatics,
Faculty of Computer Science and Systems Engineering,
Kyushu Institute of Technology, Iizuka, 820--8502, Japan.
Corresponding author: li260@bio.kyutech.ac.jp

Junjun Zheng
Department of Information Science and Engineering,
Ritsumeikan University, Kusatsu, 525--8577, Japan.
E-mail: jzheng@asl.cs.ritsumei.ac.jp

Hiroyuki Okamura
Graduate School of Advanced Science Engineering,
Hiroshima University, Higashihiroshima, 739--8527 Japan.
E-mail: okamu@hiroshima-u.ac.jp

Tadashi Dohi
Graduate School of Advanced Science Engineering,
Hiroshima University, Higashihiroshima, 739--8527 Japan.
E-mail: dohi@hiroshima-u.ac.jp

(Received on February 7, 2022; Accepted on March 18, 2022)

Abstract
With the growing demand for high reliability and safety software, software reliability prediction has attracted more and more attention to identifying potential faults in software. Software reliability growth models (SRGMs) are the most commonly used prediction models in practical software reliability engineering. However, their unrealistic assumptions and environment-dependent applicability restrict their development. Recurrent neural networks (RNNs), such as the long short-term memory (LSTM), provide an end-to-end learning method, have shown a remarkable ability in time-series forecasting and can be used to solve the above problem for software reliability prediction. In this paper, we present an attention-based encoder-decoder RNN called EDRNN to predict the number of failures in the software. More specifically, the encoder-decoder RNN estimates the cumulative faults with the fault detection time as input. The attention mechanism improves the prediction accuracy in the encoder-decoder architecture. Experimental results demonstrate that our proposed model outperforms other traditional SRGMs and neural network-based models in terms of accuracy.

Keywords- Software reliability, Recurrent neural networks (RNNs), Long short-term memory (LSTM), Encoder-decoder, Attention mechanism.

1. Introduction

With the rapid development of software technology, more and more clients use software projects to improve office efficiency. However, software failures increase rapidly as the complexity and the use of software increase. Since software failures may lead to economic and financial loss, software engineers should design more functional, safe, and reliable software systems for clients. Due to the increasing demand for high reliability and safety software, software reliability prediction becomes more vital and meaningful (Chopra et al., 2020).

In general, software reliability can be defined as the probability of the software running within a specified period. The time between successive failures or cumulative failure times is an essential indicator of software reliability (Musa et al., 1990, Wood, 1996, Nissas and Gasmi, 2021). In the past several decades, most of the existing analytical software measured and predicted software reliability using the software reliability growth models (SRGMs) (Okamura and Dohi, 2013, Okamura et al., 2013, Deepika et al. 2017, Singh et al., 2017, Datta et al., 2020). Generally, these models rely on a priori assumptions about the nature of software failures and the stochastic behavior of the software failure process (Cai et al., 1991, Park et al., 1999, Cai et al., 2001, Utkin et al., 2002, Tian and Noore, 2004). Then, SRGMs estimate parameters with the maximum likelihood estimation (Kim et al., 2015, Inoue et al., 2017) or least squares (Huang et al., 2005, Kadali et al., 2022) with the models. However, different assumptions and models make estimated parameters being not the same. Also, some unrealistic assumptions and environment-dependent applicability restrict the development of SRGMs. In other words, different models may have different predictive performances at testing phases across various projects. It is impossible to make assumptions to include all testing cases. Therefore, the above methods are costly and time-limited for software reliability prediction in realistic scenarios, and there exists no single model that can best suit all testing conditions.

Non-parametric models (Karunanithi et al., 1992, Adnan and Yaacob, 1994, Sitte, 1999, Adnan et al., 2000, Ho et al., 2003, Wang et al., 2014), such as neural networks, have been used in the last two decades to overcome the above problems. Most neural networks are one-to-one (One2One) feedforward neural networks that build the SRGMs with single input and output. Neural networks (Fu et al., 2017) provide an end-to-end learning method, where the model learns all the steps between the initial input phase and the final output result. In general, it is not necessary to make any assumptions for the neural networks-based models, and they are not influenced by any external parameters (parameter-free). It has been an alternative approach in evaluating and predicting software reliability.

Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) is a variant of recurrent neural networks (RNNs) that has an excellent ability to learn long-term dependencies of time-series data. LSTM is widely used in the field of language modeling and generating text (Roemmele and Gordon, 2018), stock price prediction (Li et al., 2019), and speech recognition (Saon et al., 2021). However, few works considered using RNNs to predict the cumulative number of detected faults for software. Encoder-decoder RNNs (Sutskever et al., 2014) are variants of RNNs and are widely applied in machine translations (Datta et al., 2020). In general, encoder-decoder RNNs mainly contain two RNNs, called the encoder and decoder, respectively. The encoder can encode sequential input into a latent vector, while the decoder decodes the latent vector into sequential output. (Wang and Zhang, 2018) is the first to predict software reliability using the encoder-decoder RNNs. They take the detected time of faults as input and take a cumulative number of detected faults as output. Also, they used the first three timestamps as training data to predict the cumulative faults of the fourth timestamp. However, since the amount of data in the fault datasets is small, overfitting may occur during the model training phase. Therefore, an RNN with few parameters is preferred for the prediction.

On the other hand, the RNN with few parameters may also lead to the insufficient predictive ability of the model, called underfitting. For example, the prediction model based on encoder-decoder LSTM proposed in (Wang and Zhang, 2018) has an underfitting problem. Therefore, it is not easy to adjust the parameters of an encoder-decoder RNN to find an optimal model for

software reliability prediction when using a small dataset. In addition, since the complex source code of software may lead to software failures, a model with better predictive ability is essential to improve work efficiency and reduce economic losses. We propose an encoder-decoder RNN with an attention mechanism to address the above issues. Specifically, the encoder-decoder RNN estimates the cumulative faults with the fault detection time as input. The attention mechanism improves the prediction accuracy in the encoder-decoder architecture. The test dataset is split from a benchmark fault dataset to verify our proposed model. For brevity, the main contributions of this paper are as follows:

- **Architecture design:** An encoder-decoder RNN is presented to predict the cumulative fault numbers in software and assess the software reliability with the fault detection time as input. Unlike the traditional SRGMs, the encoder-decoder RNN is a parameter-free model without extra assumptions.
- **Attention mechanism:** The attention mechanism can extract more significant features from the hidden layers of encoder-decoder RNNs. The attention mechanism-based model may improve the predictive ability when the dataset is small. Therefore, the attention mechanism is taken into account.
- **Performance improvement:** Traditional SRGMs and neural networks as the baseline models are compared with the attention-based encoder-decoder RNN to validate the effectiveness in experiments.

The remainder of the paper is organized as follows. In section 2, we review some related research works. Section 3 introduces the proposed attention-based encoder-decoder RNNs for software reliability prediction. Then, numerical experiments with a benchmark fault dataset are conducted to assess software reliability in section 4. Finally, we conclude the paper with remarks in section 5.

2. Related Works

We briefly introduce some related works on using software reliability growth models and neural networks in software reliability modeling and prediction.

2.1 Software Reliability Growth Models

The SRGMs aim to specify a stochastic process that describes the software behavior for software failures, which can estimate reliability, measure current states, and predict the number of faults (Amin et al., 2013). Most of the SRGMs assume that the software system is modified and corrected as the test processes to reduce failure rates and improve reliability. Non-homogeneous Poisson process (NHPP)-based SRGMs have attracted more and more attention since they can represent the failure process under various conditions (He, 2013, Li and Pham, 2017, Li and Pham, 2019, Song et al., 2019). Failure time distributions have been applied to SRGMs since they can characterize most NHPP-based SRGMs. Failure time distributions such as Goel and Okumoto model (Goel, 1979), generalized delayed S-shaped model (Yamada et al., 1983, Zhao and Xie, 1996), modified Duane model (Littlewood, 1984), inflection S-shaped model (Ohba, 1984), and Goel (Weibull) model (Goel, 1985) was used in the NHPP-based SRGMs. To better understand the failure time distributions and their corresponding NHPP-based SRGMs, we demonstrate 11 commonly used mean value functions in Table 1.

Table 1. The commonly used NHPP-based SRGMs.

Description	Mean Value Functrion
Exponential distribution (Goel and Okumoto, (1979)	$\Lambda(t) = aF(t), F(t) = 1 - e^{-bt}$
Gamma distribution (Yamada et al., 1983, Yamada and Osaki, 1985)	$\Lambda(t) = aF(t), F(t) = \int_0^t \frac{c^b s^{b-1} e^{-cs}}{\Gamma(b)} ds$
Pareto type-II distribution (Littlewood, 1984, Abdel-Ghaly et al., 1986)	$\Lambda(t) = aF(t), F(t) = 1 - (\frac{c}{t+c})^b$
Truncated normal distribution (Okamura et al., 2013)	$\Lambda(t) = a\frac{F(t) - F(0)}{1 - F(0)}, F(t) = \frac{1}{\sqrt{2\pi b}} \int_0^t e^{-\frac{(s-c)^2}{2b^2}} ds$
Log-normal distribution (Okamura et al., 2013, Achcar et al., 1998)	$\Lambda(t) = aF(\log t), F(t) = \frac{1}{\sqrt{2\pi b}} \int_{-\infty}^t e^{-\frac{(s-c)^2}{2b^2}} ds$
Truncated logistic distribution (Ohba, 1984)	$\Lambda(t) = a\frac{F(t) - F(0)}{1 - F(0)}, F(t) = \frac{1}{1 + e^{-\frac{t-c}{b}}}$
Log-logistic distribution (Gokhale and Trivedi, 1998)	$\Lambda(t) = aF(\log t), F(t) = \frac{1}{1 + e^{-\frac{t-c}{b}}}$
Truncated extreme-value distribution (max) (Ohishi et al., 2009, Yamada, 1992)	$\Lambda(t) = a\frac{F(t) - F(0)}{1 - F(0)}, F(t) = \exp(-\exp(-\frac{t-c}{b}))$
Log-extreme-value distribution (max) (Ohishi et al., 2009)	$\Lambda(t) = aF(\log t), F(t) = \exp(-\exp(-\frac{t-c}{b}))$
Truncated extreme-value distribution (min) (Ohishi et al., 2009)	$\Lambda(t) = a\frac{F(0) - F(-t)}{F(0)}, F(t) = \exp(-\exp(-\frac{t-c}{b}))$
Log-extreme-value distribution (min) (Ohishi et al., 2009, Goel, 1985)	$\Lambda(t) = a(1 - F(-\log t)), F(t) = \exp(-\exp(-\frac{t-c}{b}))$

$F(t)$ and $\Lambda(t)$ represent the cumulative distribution function and the mean value function of fault-detection time. In general, specific assumptions are necessary for these SRGMs. However, some of them are questionable and unrealistic. For example, most SRGMs assume that the successive failure times are independent of each other. In practice, the test case of functional testing is dependent. Therefore, the unrealistic and questionable assumptions are not suitable for a general situation.

2.2 Neural Network-based Models

In recent years, some literature has used neural networks for software reliability prediction since the neural network approaches have a powerful ability to approximate any linear and non-linear continuous function. (Karunanithi et al., 1991) applied neural networks to predict the cumulative number of failures. They used feed-forward neural networks-based software reliability models in their works. (Hu et al., 2006) applied deep neural networks to improve the early reliability prediction for current projects/releases by reusing the failure data from past projects/releases. The results showed that their work has better results than the traditional methods. However, the artificial neural network did not take time series into account. (Karunanithi et al. 1992) used a simple feed-forward network, simple recurrent networks trained using a standard back-propagation algorithm, and a semi-recurrent network to predict software reliability. They also calculated the average error (AE) for endpoint and next-step prediction. Although they took the effect of time series into account, the recurrent neural networks had the problem of gradient vanishing. Figure 1 shows the structure of the RNN. The gray and blank circles represent input and a hidden state, respectively. The input data $\boldsymbol{x}_1$ at the time step t_1 is fed into the RNN. Then, the features of $\boldsymbol{x}_1$ are extracted and stored in $\boldsymbol{h}_1$. For the next time step, the input data $\boldsymbol{x}_2$ and the hidden state $\boldsymbol{h}_2$ are the input to calculate the hidden state $\boldsymbol{h}_2$. (Su and Huang, 2007) proposed a dynamic weighted combinational model (DWCM). They trained the model with different activation functions, such as the sigmoid and Tanh function. They validated their model on two

datasets and compared their proposed model with statistical models. Experimental results showed that the proposed dynamic weighted combinational model significantly gave better predictions. Similarly, the proposed model is based on the artificial neural network, which has been proven to lack the ability to process time-series data.

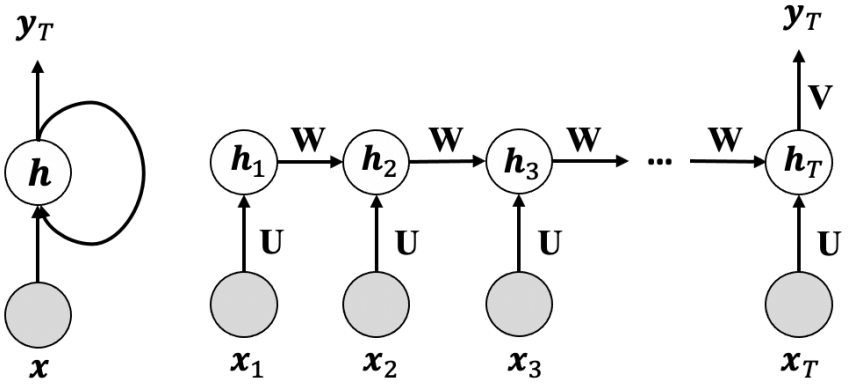

Figure 1. An example of the RNN model.

The above literature has demonstrated that it is possible to apply neural networks (i.e., artificial and recurrent neural networks) for software reliability prediction. Especially for RNNs, the prediction can be treated as a sequence-to-sequence translation problem. RNNs can effectively alleviate the long-term dependence problem, but it is powerless to the unequal length of input and output sequences problem. Recently, encoder-decoder RNNs have shown the power in solving the problem of unequal length. (Wang and Zhang, 2018) is the first work to use a deep encoder-decoder model to predict the number of software faults. Unlike the traditional RNNs, they applied two variants of RNNs called LSTMs, as the encoder and decoder. Then, they predicted the cumulative fault number for both the next step and endpoint prediction. The prediction results on 14 datasets showed that the proposed model worked well. However, all the datasets are too small, which may cause an overfitting problem during the training phase. On the other hand, the attention mechanism can effectively improve the model's accuracy. In this paper, the fault detection time is considered as an input time series and regard the cumulative number of detected faults as an output sequence by taking the accuracy into account.

3. Model Description

We mainly introduce the LSTM neural networks, encoder-decoder RNNs, and our proposed attention-based encoder-decoder RNN (EDRNN) in detail.

3.1 Long Short-term Memory

Figure 2 demonstrates the structure of an LSTM neural network. In general, an LSTM maintains three gates, called the input gate (i in Figure 2), forget gate (f in Figure 2), and output gate (o in Figure 2).

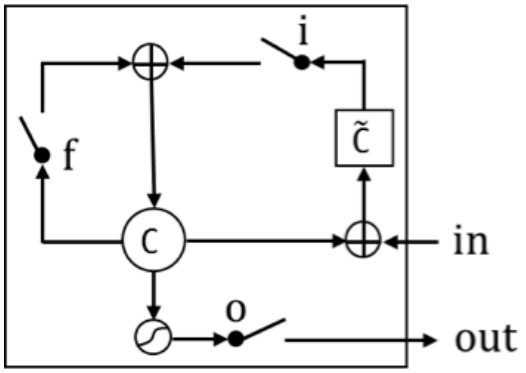

Figure 2. An example of the LSTM cell.

Formally, given the detected time of faults as input sequence $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T\}$. An LSTM extracts long and short-term dependencies of the time series by the following composite functions:

$$\boldsymbol{i}_t = \delta(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ci}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i) \quad (1)$$

$$\boldsymbol{f}_t = \delta\left(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cf}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f\right) \quad (2)$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t\boldsymbol{c}_{t-1} + \boldsymbol{i}_t\tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \quad (3)$$

$$\boldsymbol{o}_t = \delta(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{co}\boldsymbol{c}_{t-1} + \boldsymbol{b}_o) \quad (4)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t\tanh(\boldsymbol{c}_t) \quad (5)$$

where $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{c}_t$ are the input gate, forget gate, output gate and cell content, respectively. δ and tanh are activation functions, and $\boldsymbol{W}$ and $\boldsymbol{b}$ are weight matrices and bias, respectively. $\boldsymbol{h}_t$ represents the hidden state of current step t, which saves the extracted features of the time series $\boldsymbol{X}$. Hence the name, the input gate and forget gate work as inputs to the cell state $\boldsymbol{c}_t$. Intuitively, the input gate decides what information is relevant to add from the current state. The forget gate decides what is relevant to keep from previous steps. The hidden state of the previous time step $\boldsymbol{h}_{t-1}$ and current input $\boldsymbol{x}_t$ is passed into an activation function to decide which values are used to update (Eq. (1)). Also, the information from the current input $\boldsymbol{x}_t$ and previous hidden state $\boldsymbol{h}_{t-1}$ is passed through an activation function to decide which values need to be forgotten (Eq. (2)). Note that the input gate and forget gate are between 0 and 1, indicating the proportion of information that needs to be updated and forgotten. Then, $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ are passed into the tanh activation function to regulate the network. Next, the input gate updates the cell content $\boldsymbol{c}_t$ (Eq. (3)). The output gate decides what information the next hidden state carries (Eq. (4)). The output gate and the current cell state are the input to decide the hidden state at the current time step (Eq. (6)). Finally, the hidden state $\boldsymbol{h}_t$ is connected by a fully connected artificial neural network with only one neuron, which can output the predicted results at time step t by applying the softmax function. The formula is demonstrated as follows:

$$\text{prediction}_t = \text{softmax}(\text{FNN}(\boldsymbol{h}_t)) \quad (6)$$

where FNN represents the fully connected neural network with only one neuron. For a better understanding, Figure 3 demonstrates the LSTM cell.

3.2 Encoder-Decoder RNN

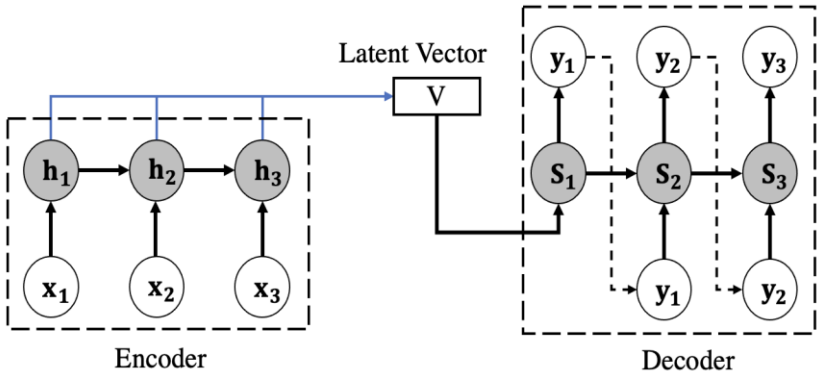

Figure 3. Structure of an encoder-decoder RNN.

An encoder-decoder RNN is composed of two RNNs and a latent vector. In this paper, each of the RNN is an LSTM neural network. Figure 3 shows the structure of an encoder-decoder RNN. The hidden state $\boldsymbol{h}_t$ $(1 \leq t \leq T)$ of the encoder at each time step is saved in a latent vector $\boldsymbol{V}$, as shown in the blue part of Figure 2. The formula is as follows:

$$\boldsymbol{V} = f(\boldsymbol{h}_1, \boldsymbol{h}_2, \dots, \boldsymbol{h}_T) \tag{7}$$

where f represents nonlinear function. Similar to the encoder, the decoder decodes the latent vector to output sequences. Specifically, the decoder predicts the next output y_{t+1} according to $\boldsymbol{V}$ and $\{y_1, y_2, \dots, y_t\}$ as follows:

$$P(y_{t+1}|\{y_1, y_2, \dots, y_t\},\ \boldsymbol{V}) = \text{Decoder}(y_t, \boldsymbol{s}_{t+1}, \boldsymbol{V}) \tag{8}$$

where P and $\boldsymbol{s}_t$ denote a probability and the hidden state of the decoder at $t+1$ time step. In Eq. (8), all the weights of $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \dots, \boldsymbol{h}_T\}$ to the latent vector $\boldsymbol{V}$ are considered the same.

3.3 Attention-based Encoder-decoder RNN

Figure 4 illustrates the structure of the our proposed EDRNN model. The model includes four aspects: an encoder, a latent vector, a decoder, and an attention mechanism.

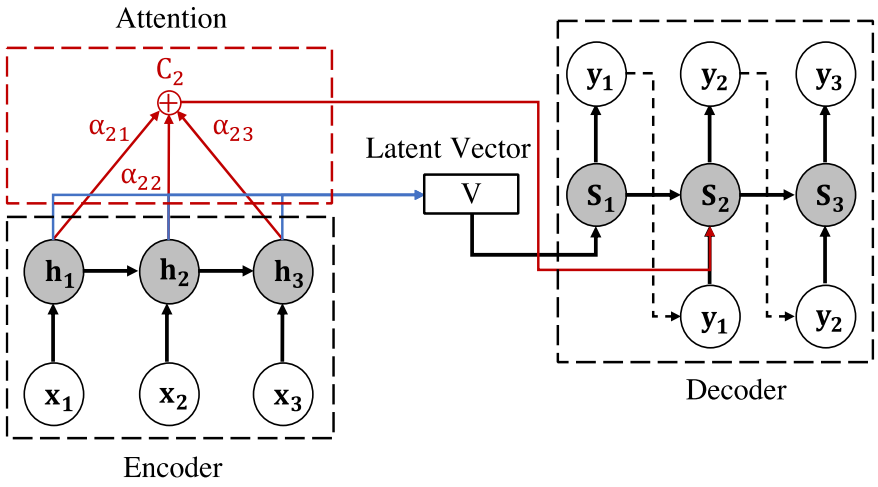

Figure 4. Structure of the encoder-decoder RNNs with attention mechanism.

In general, encoder-decoder models have limitations of encoding the input sequence to one fixed-length vector and decoding each output time step. This issue may make it more challenging for neural networks to cope with long sequences.

Instead of encoding a sequence to a fixed-length vector, the attention mechanism-based neural networks can assign different weights $\boldsymbol{h}_t$ to the latent vector $\boldsymbol{V}$. Therefore, the attention is taken into account, as shown in the red part of Figure 4. Formally, the formulas of the attention mechanism is demonstrated as below:

$$e_{ti} = \boldsymbol{s}_t \boldsymbol{W}_{sh} \boldsymbol{h}_i \tag{9}$$

$$\alpha_{ti} = \text{softmax}(e_{ti}) = \frac{\exp(e_{ti})}{\sum_{j=1}^{T_d} \exp(e_{tj})} \tag{10}$$

$$\boldsymbol{c}_t = \sum_{i=1}^{T_d} \alpha_{ti} \boldsymbol{h}_i \tag{11}$$

$$\boldsymbol{s}_t = \text{LSTM}(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t) \tag{12}$$

Where T_d denotes the decoder sequence length. The attention mechanism can help the encoder-decoder RNN model assign different weights to the latent vector and predict the cumulative number of faults.

4. Experiments

In this section, several extensive experiments are conducted to validate the effectiveness of our proposed model.

4.1 Dataset

A benchmark fault dataset is selected to predict the cumulative fault number. The dataset is collected from a real-time control application with approximately 870,000 code lines in (Tohma et al., 1991). Table 2 demonstrates the dataset, where $\boldsymbol{x}_{(i)}$ and $\boldsymbol{c}_{(i)}$ represent the observed detected

faults and cumulative number of observed detected faults, respectively. The dataset is divided into a training set and testing set to examine this dataset: the first 100 sets of data are used for training and the last six sets of data are used for testing. Figure 5 shows the train and test data of the fault dataset.

Table 2. The benchmark fault dataset.

Day	$X(i)$	$C(i)$	Day	$X(i)$	$C(i)$	Day	$X(i)$	$C(i)$	Day	$X(i)$	$C(i)$
1	4	4	29	6	218	57	3	458	85	1	523
2	0	4	30	6	224	58	6	464	86	1	524
3	7	11	31	4	228	59	6	470	87	0	524
4	10	21	32	12	240	60	3	473	88	2	526
5	13	34	33	6	246	61	0	473	89	0	526
6	8	42	34	7	253	62	0	473	90	0	526
7	13	55	35	8	261	63	3	476	91	1	527
8	4	59	36	11	272	64	0	476	92	1	528
9	7	66	37	6	278	65	4	480	93	0	528
10	8	74	38	9	287	66	0	480	94	0	528
11	1	75	39	7	294	67	1	481	95	0	528
12	6	81	40	12	306	68	2	483	96	0	528
13	13	94	41	12	318	69	0	483	97	1	529
14	7	101	42	15	333	70	1	484	98	0	529
15	9	110	43	14	347	71	2	486	99	1	530
16	8	118	44	7	354	72	5	491	100	0	530
17	5	123	45	9	363	73	3	494	101	0	530
18	10	133	46	11	374	74	2	496	102	0	530
19	7	140	447	5	379	75	1	497	103	0	530
20	11	151	8	7	386	76	11	508	104	2	532
21	5	156	49	7	393	77	1	509	105	0	532
22	8	164	50	14	407	78	0	509	106	1	533
23	13	177	51	13	420	79	2	511	107	0	533
24	9	186	52	14	434	80	2	513	108	2	535
25	7	193	53	11	445	81	4	517	109	0	535
26	7	200	54	2	447	82	1	518	-	-	-
27	5	205	55	4	451	83	0	518	-	-	-
28	7	212	56	4	455	84	4	522	-	-	-

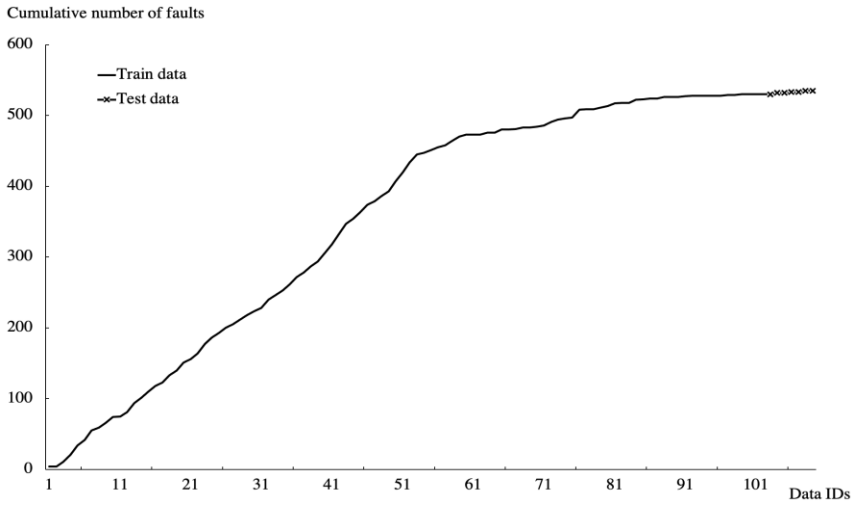

Figure 5. Training data and test data of the fault dataset.

4.2 Experimental Configuration

All the initial weights $\boldsymbol{W}$ and the bias $\boldsymbol{b}$ are set randomly. We use Keras 2.4.3 deep learning library and program with Python 3.5.6 for the implementation.
The experimental environment is demonstrated as follows:

- **CPU:** 3.6 GHz 10 core Intel Core i9
- **Memory:** 64GB 2667 MHz DDR4
- **OS:** macOS Big Sur

4.3 Baseline Models

11 SRGMs and four neural networks are compared to the proposed EDRNN model. The details are shown in Table 3.

Table 3. Description of the baseline models.

Model	Description
Exponential	Exponential distribution
Gamma	Gamma distribution
Pareto	Pareto type-II distribution
Truncated Normal	Truncated normal distribution
Log Normal	Log-normal distribution
Truncated Logistic	Truncated logistic distribution
Log logistic	Log-logistic distribution
Truncated Extreme Value Maximum	Truncated extreme-value distribution (max)
Log Extreme Value Maximum	Log-extreme-value distribution (max)
Truncated Extreme Value Minimum	Truncated extreme-value distribution (min)
Log Extreme Value Minimum	Log-extreme-value distribution (min)
LSTM (Seq2One)	Sequence-to-one via LSTM
LSTM (Seq2Seq)	Sequence-to-sequence via LSTM
EDNNs (Seq2One)	Sequence-to-one via LSTM encoder-decoder
EDNNs (Seq2Seq)	Sequence-to-sequence via LSTM encoder-decoder
EDRNN	Attention-based Sequence-to-sequence via LSTM encoder-decoder

Mean squared error (MSE) is calculated to evaluate the prediction results of the baseline models and our proposed EDRNN model. The LSTM and the EDNNs can be divided into two cases: sequence-to-sequence (Seq2Seq) and sequence-to-one (Seq2One), respectively. The case of Seq2Seq takes a sequence as the input of the encoder and outputs a sequence of the decoder. The Seq2One takes a sequence of fault detected time as the input and outputs the cumulative number of software failure occurrences.

4.4 Comparison with SRGMs

Figure 6 demonstrates the predicted results of 11 commonly used SRGMs and our EDRNN model. X and Y-axis represent the ID and the cumulative number of faults of the six sets. From the figure, we observe that the predicted results of the truncated normal distribution and truncated extreme-value distribution (min) are the same (i.e., 529). In other words, the two models cannot predict the number of faults well. On the other hand, the results of the exponential distribution, Pareto type-II distribution, and log-normal distribution are the same. Although the values increased with increasing timestamps, the difference from the test data is getting larger. The difference between the test data and the results of the log-extreme-value distribution (max) is the largest among the 11 models.

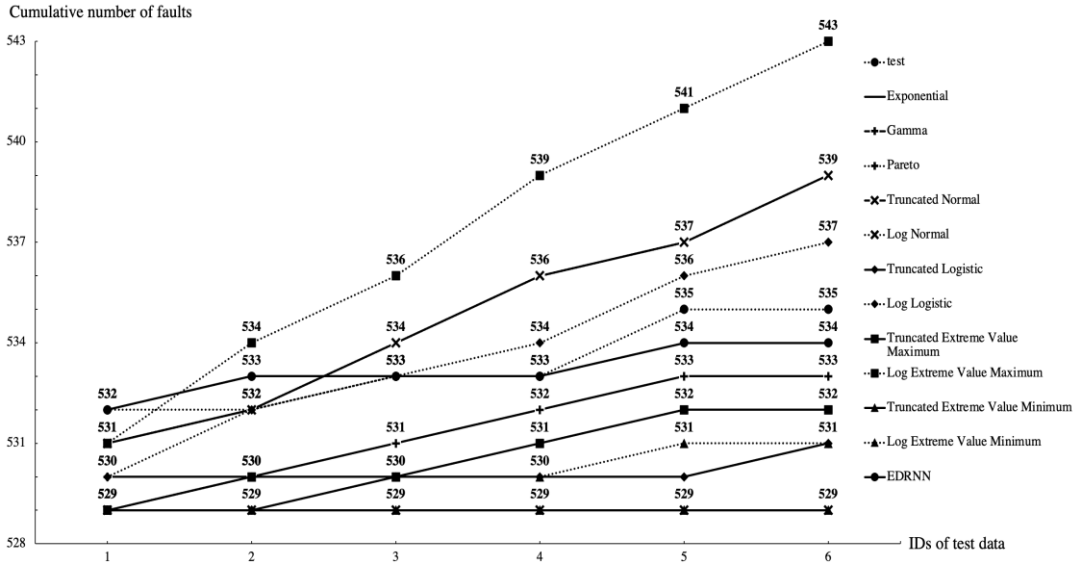

Figure 6. Predicted results of the proposed model (EDRNN) compared with 11 SRGMs.

Table 4 evaluated the MSE values of the 11 SRGMs. We find that the predictions of the log-logistic distribution have the smallest MSE in the 11 models and the MSE is 1.70. Our proposed EDRNN does not need any assumption and the MSE (0.50) is much smaller than the log-logistic distribution. In other words, EDRNN works better than the traditional SRGMs. Table 4. MSE results of the proposed model and the 11 SRGMs.

Table 4. MSE results of the proposed model and 11 SRGMs.

Model	MSE
Exponential	5.20
Gamma	3.50
Pareto	5.20
Truncated Normal	20.3
Log Normal	5.20
Truncated Logistic	12.8
Log Logistic	1.70
Truncated Extreme Value Maximum	7.30
Log Extreme Value Maximum	25.0
Truncated Extreme Value Minimum	20.3
Log Extreme Value Minimum	10.5
EDRNN	0.50

4.5 Comparison with Neural Networks

To further validate our proposed model, we also compare our model with neural network-based models. Figure 7 demonstrates the predicted results of two LSTMs, two encoder-decoder RNNs, and our EDRNN model. We find that the prediction results of the LSTM(Seq2Seq), LSTM(Seq2One), and EDNNs (Seq2One) are quite different from the test data. The prediction results of the EDNNs (Seq2Seq) and EDRNN are very close to the test data. We calculated the MSE values to qualify the prediction results, which can be seen in Table 5.

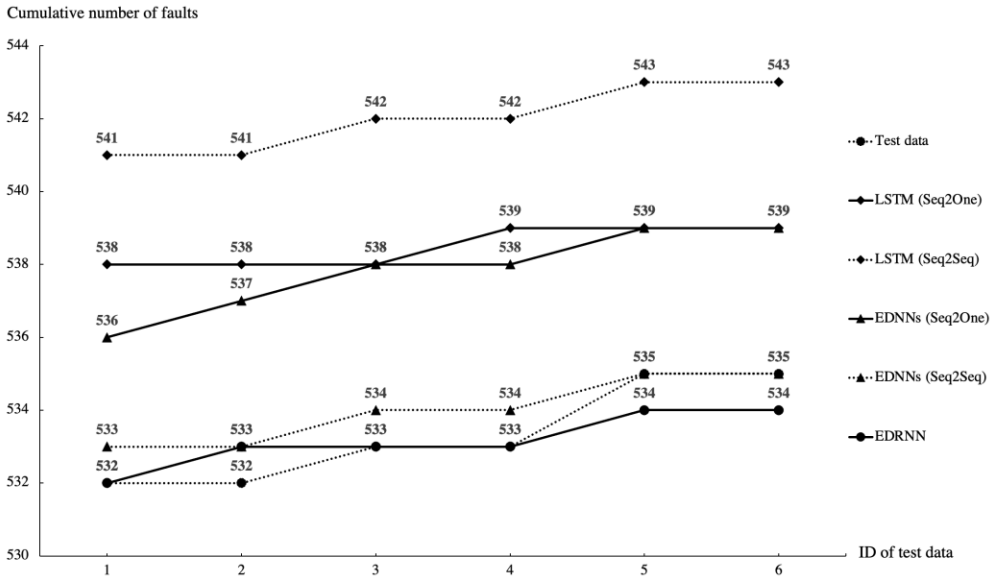

Figure 7. Predicted results of the proposed model (EDRNN) compared with neural network-based models.

Table 5. MSE results of the proposed model and neural network-based models.

Model	MSE
LSTM (Seq2One)	27.5
LSTM (Seq2Seq)	75.3
EDNNs (Seq2One)	20.5
EDNNs (Seq2Seq)	0.67
EDRNN	0.50

From the table, we can see that the MSE values of the EDNNs(Seq2Seq) (0.67) and our proposed EDRNN (0.50) are much smaller than other baseline models. Unlike the EDNNs(Seq2Seq) model, our proposed EDRNN took the attention mechanism into account. Because our attention mechanism can extract more features of the fault dataset than the conventional neural networks models, the MSE values of the predicted results are the smallest in all baseline models. Therefore, our proposed EDRNN can work well on the small dataset and the attention mechanism can improve the accuracy of the prediction results.

5. Conclusion

Software reliability has been regarded as a commonly used factor to quantify software faults. In practice, SRGMs are the widely used models to estimate or predict the cumulative fault number in software. However, some unrealistic assumptions are indispensable when using such SRGMs for prediction. These shortcomings limit the scalability and practicality of SRGMs in practical problems. In this paper, we proposed attention-based encoder-decoder RNNs for predicting the cumulative fault number in software and assessing the software reliability. The encoder-decoder RNN models can overcome the shortcomings of traditional SRGMs and make predictions effectively on different types of datasets after training. An encoder-decoder RNN can encode the input information and store the abstract information of the encoder in a fixed-dimensional vector. For the vector, each vector element has the same weight of 1 by default. But in practical problems, different features should have different weights. Therefore, an attention mechanism between the

encoder and the decoder layer is applied to improve the prediction accuracy. In experiments, 11 traditional SRGMs and neural network-based models such as LSTM and encoder-decoder model were the baseline models to compare with our proposed EDRNN. Experiment results validated the effectiveness of our proposed EDRNN and verified that the EDRNN improved the accuracy of prediction in software reliability.

One of the shortcomings of this paper is the lack of validation of our model with different fault datasets. We have not verified the effectiveness and impact of the proposed EDRNN in different datasets. Since different types of faults datasets may have a different effect on the prediction, we would like to apply more types and numbers of fault datasets to validate our proposed model in the future. After solving these problems, we believe that our proposed EDRNN model will help software engineers to predict software faults in advance in the software development and testing stage to reduce unnecessary losses.

Conflict of Interest

The authors confirm that there is no conflict of interest to declare for this publication.

Acknowledgements

The authors would like to express their sincere thanks to the editor and anonymous reviewers for their time and valuable suggestions.

References

Abdel-Ghaly, A. A., Chan, P. Y., & Littlewood, B. (1986). Evaluation of competing software reliability predictions. *IEEE Transactions on Software Engineering*, (*9*), pp. 950−967.

Achcar, J. A., Dey, D. K., & Niverthi, M. (1998). A Bayesian approach using nonhomogeneous poisson processes for software reliability models. In *Frontiers in Reliability* (pp. 1−18). https://doi.org/10.1142/9789812816580_0001

Adnan, W. A., & Yaacob, M. H. (1994, December). An integrated neural-fuzzy system of software reliability prediction. In *Proceedings of 1994 1st International Conference on Software Testing, Reliability and Quality Assurance (STRQA'94)* (pp. 154−158). IEEE. New Delhi, India.

Adnan, W. A., Yaakob, M., Anas, R., & Tamjis, M. R. (2000, September). Artificial neural network for software reliability assessment. In *2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No. 00CH37119)* (*Vol. 3*, pp. 446−451). IEEE. Kuala Lumpur, Malaysia.

Amin, A., Grunske, L., & Colman, A. (2013). An approach to software reliability prediction based on time series modeling. *Journal of Systems and Software*, *86*(*7*), pp. 1923−1932.

Cai, K. Y., Cai, L., Wang, W. D., Yu, Z. Y., & Zhang, D. (2001). On the neural network approach in software reliability modeling. *Journal of Systems and Software*, *58*(*1*), pp. 47−62.

Cai, K. Y., Wen, C. Y., & Zhang, M. L. (1991). A critical review on software reliability modeling. *Reliability Engineering & System Safety*, *32*(*3*), pp. 357−371.

Chopra, S., Nautiyal, L., Malik, P., Ram, M., & Sharma, M. K. (2020). A non-parametric approach for survival analysis of component-based software. *International Journal of Mathematical, Engineering and Management Sciences*, *5*(*2*), pp. 309−318.

Datta, D., David, P. E., Mittal, D., & Jain, A. (2020). Neural machine translation using recurrent neural network. *International Journal of Engineering and Advanced Technology*, *9(4)*, pp. 1395−1400.

Deepika, O. S., Anand, A., & Singh, J. N. (2017). Testing domain dependent software reliability growth models. *International Journal of Mathematical, Engineering and Management Sciences*, *2*(*3*), pp. 140−149.

Goel, A. L., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, *28*(*3*), pp. 206−211.

Goel, A. L. (1985). Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering*, (*12*), pp. 1411−1423.

Gokhale, S. S., & Trivedi, K. S. (1998, November). Log-logistic software reliability growth model. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium* (Cat. No. 98EX231) (pp. 34−41). IEEE. Washington, DC, USA.

He, Y. (2013, May). NHPP software reliability growth model incorporating fault detection and debugging. In *2013 IEEE 4th International Conference on Software Engineering and Service Science* (pp. 225−228). IEEE. Beijing, China.

Ho, S. L., Xie, M., & Goh, T. N. (2003). A study of the connectionist models for software reliability prediction. *Computers & Mathematics with Applications*, *46*(*7*), pp. 1037−1045.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(*8*), pp. 1735−1780.

Hu, Q. P., Dai, Y. S., Xie, M., & Ng, S. H. (2006, September). Early software reliability prediction with extended ANN model. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)* (Vol. 2, pp. 234−239). IEEE. Chicaco, USA.

Huang, C. Y., & Lyu, M. R. (2005). Optimal release time for software systems considering cost, testing-effort, and test efficiency. *IEEE Transactions on Reliability*, *54*(*4*), pp. 583−591.

Inoue, S., Hotta, K., & Yamada, S. (2017). On estimation of number of detectable software faults under budget constraint. *International Journal of Mathematical, Engineering and Management Sciences*, *2*(*3*), pp. 135−139.

Kadali, D. K., Naik, M. C., & Mohan, R. J. (2022). Estimation of data parameters using cluster optimization (*No. 7293*). EasyChair.

Karunanithi, N., Malaiya, Y. K., & Whitley, L. D. (1991, May). Prediction of software reliability using neural networks. *In ISSRE* (pp. 124−130).

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, *18*(*7*), pp. 563−574.

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Using neural networks in reliability prediction. *IEEE Software*, *9(4)*, pp. 53−59.

Kim, T., Lee, K., & Baik, J. (2015). An effective approach to estimating the parameters of software reliability growth models using a real-valued genetic algorithm. *Journal of Systems and Software*, *102*, pp. 134−144.

Li, C., Zhang, X., Qaosar, M., Ahmed, S., Alam, K.M.R., & Morimoto, Y. (2019). Multi-factor-based stock price prediction using hybrid neural networks with attention mechanism," *2019 IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 961−966.

Li, Q., & Pham, H. (2017). NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. *Applied Mathematical Modelling*, *51*, pp. 68−85.

Li, Q., & Pham, H. (2019). A generalized software reliability growth model with consideration of the uncertainty of operating environments. *IEEE Access*, *7*, pp. 84253−84267.

Littlewood, B. (1984). Rationale for a modified Duane model. *IEEE Transactions on Reliability*, *33*(*2*), pp. 157−159.

Musa, J.D., Iannino, A., & Okumoto, K. (1990). Software reliability. *Advances in Computers*, *30*, pp. 85−170.

Nissas, W., & Gasmi, S. (2021). On the maintenance modeling of a hybrid model with exponential repair efficiency. *International Journal of Mathematical, Engineering and Management Sciences*, *6*(*1*), pp. 254−267.

Ohba, M. (1984). Software reliability analysis models. *IBM Journal of Research and Development*, *28*(*4*), pp. 428−443.

Ohba, M. (1984). Inflection S-shaped software reliability growth model. *In Stochastic Models in Reliability Theory* (pp. 144−162). Springer, Berlin, Heidelberg.

Ohishi, K., Okamura, H., & Dohi, T. (2009). Gompertz software reliability model: Estimation algorithm and empirical validation. *Journal of Systems and Software*, *82*(*3*), pp. 535−543.

Okamura, H., & Dohi, T. (2013, November). SRATS: Software reliability assessment tool on spreadsheet (Experience report). *In 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 100−107). IEEE. Pasadena, CA, USA.

Okamura, H., Dohi, T., & Osaki, S. (2013). Software reliability growth models with normal failure time distributions. *Reliability Engineering & System Safety*, *116*, pp. 135−141.

Park, J. Y., Lee, S. U., & Park, J. H. (1999). Neural network modeling for software reliability prediction from failure time data. *Journal of Electrical Engineering and Information Science*, *4*(*4*), pp. 533−538.

Roemmele, M., & Gordon, A. S. (2018, March). Automated assistance for creative writing with an rnn language model. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion* (pp. 1−2). https://doi.org/10.1145/3180308.3180329

Saon, G., Tüske, Z., Bolanos, D., & Kingsbury, B. (2021, June). Advancing RNN transducer technology for speech recognition. In ICASSP 2021-2021 *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5654−5658). IEEE. Toronto, ON, Canada.

Singh, J., Bhati, S., Prasanan, A. R., & Vayas, A. (2017). Stochastic formulation of fault severity based multi release SRGM using the effect of logistic learning. *International Journal of Mathematical, Engineering and Management Sciences*, *2*(*3*), pp. 172−184.

Sitte, R. (1999). Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration. *IEEE Transactions on Reliability*, *48*(*3*), pp. 285 291.

Song, K. Y., Chang, I. H., & Pham, H. (2019). NHPP software reliability model with inflection factor of the fault detection rate considering the uncertainty of software operating environments and predictive analysis. *Symmetry*, *11*(*4*), p. 521.

Su, Y. S., & Huang, C. Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software*, *80*(*4*), pp. 606−615.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, p. 27.

Tian, L., & Noore, A. (2004). Software reliability prediction using recurrent neural network with Bayesian regularization. *International Journal of Neural Systems*, *14*(03), pp. 165−174.

Tohma, Y., Yamano, H., Ohba, M., & Jacoby, R. (1991, January). Parameter estimation of the hyper-geometric distribution model for real test/debug data. In *Proceedings. 1991 International Symposium on Software Reliability Engineering* (pp. 28−34). IEEE Computer Society.

Utkin, L. V., Gurov, S. V., & Shubinsky, M. I. (2002). A fuzzy software reliability model with multiple-error introduction and removal. *International Journal of Reliability, Quality and Safety Engineering*, *9(03)*, pp. 215−227.

Wang, J., Wu, Z., Shu, Y., Zhang, Z., & Xue, L. (2014, July). A study on software reliability prediction based on triple exponential smoothing method (WIP). In *Proceedings of the 2014 Summer Simulation Multiconference* (pp. 1−9).

Wang, J., & Zhang, C. (2018). Software reliability prediction using a deep learning model based on the RNN encoder–decoder. *Reliability Engineering & System Safety*, *170*, pp. 73−82.

Wood, A. (1996). Predicting software reliability. *Computer*, *29*(*11*), pp. 69−77.

Yamada, S., Ohba, M., & Osaki, S. (1983). S-shaped reliability growth modeling for software error detection. *IEEE Transactions on reliability*, *32*(*5*), pp. 475−484.

Yamada, S., & Osaki, S. (1985). Software reliability growth modeling: Models and applications. *IEEE Transactions on software engineering*, (*12*), pp. 1431−1437.

Yamada, S. (1992). A stochastic software reliability growth model with Gompertz curve. *Journal of Information Processing*, *15*(*3*), p. 495.

Yangzhen, F., Hong, Z., Chenchen, Z., & Chao, F. (2017, July). A software reliability prediction model: using improved long short term memory network. In 2017 *IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (pp. 614−615). IEEE. Prague, Czech Republic.

Zhao, M., & Xie, M. (1996). On maximum likelihood estimation for a general non-homogeneous Poisson process. *Scandinavian Journal of Statistics*, *23*, pp. 597−607.