Optimized Long Short-Term Memory Deep Learning Approach for Precise Software Reliability Fault Estimation

Juwon Jung[1], and Chaebong Sohn[2]

[1]Department of Defense Industry AI & Robot Convergence, Kwangwoon University and MOASOFT Co., Ltd
[2]Department of Defense Acquisition Program, Kwangwoon University, 20, Gwangun-ro, Nowon-gu, seoul, Republic of Korea

wopalw@gmail.com[1], cbsohn@kw.ac.kr[2]

Abstract

This study proposes an LSTM (Long Short-Term Memory)-based deep learning model to overcome the concerns of existing Software Reliability Growth Models (SRGM), which suffer from reduced prediction accuracy due to limited fault data in the early software testing environment. The study involves generating synthetic fault data for the software testing phase, optimizing model performance through hyperparameter tuning with Bayesian Optimization, and applying GPU memory optimization and a sliding window technique. The results demonstrate that the proposed LSTM model outperforms traditional SRGM in prediction accuracy. In the 1–60 time interval, the RMSE of the LSTM model is approximately 37, which is 214 lower than the RMSE of 251 observed for SRGM. Notably, the LSTM model exhibited stable performance in both the early stages with insufficient data and in long-term predictions beyond 61 time intervals. The LSTM-based deep learning model presented in this study offers a novel alternative for software reliability prediction by deeply learning complex long-term patterns in time-series data and combining this capability with optimization methodologies that explore global minimum. This approach enhances the model's generalization performance and ensures consistent results under various initial conditions. Such a technical approach is expected to be highly applicable in advanced industries that demand high-reliability systems, such as defense, aerospace, and healthcare.

Keywords: *LSTM (Long Short-Term Memory), Prediction Techniques, Software Reliability Growth Models (SRGM), Reliability Assessment, Time Series Analysis*

1. Introduction

The technique of reliability prediction plays an important role in the repair of reliability during the programme's design process.[1] Conventional Software Reliability Growth Models (SRGM) employ a statistical approach to predict and estimate future fault occurrences based on historical defect data.[2] However, SRGM has concerns in prediction accuracy when the quantity or quality of data is low.[3,4] In such situations, the performance of SRGM is limited, presenting a significant challenge in software reliability prediction.

This study proposes an LSTM (Long Short-Term Memory)-based deep learning model to overcome the concerns of existing Software Reliability Growth Models (SRGM), which suffer from reduced prediction accuracy due to limited fault data in the early software testing environment. LSTM, a type of Recurrent Neural Network (RNN), has the advantage of effectively learning long-term time series data, enabling it to learn and predict long-term patterns of software fault occurrences.[5] This research focuses on the LSTM algorithm rather than comparing various models [6,7]. Additionally, by directly verifying the consistency between LSTM's prediction results and actual data distributions across different time intervals, we aim to provide deep insights into how LSTM models can be utilized for software reliability prediction.

For model optimization, we employ hyperparameter optimization and sliding window techniques to maximize model performance by targeting the global minimum, expecting to overcome the concerns of SRGM. This study uses synthetic fault data from the software testing phase to train the LSTM-based deep learning model and compares its performance with existing SRGM to demonstrate a more stable and accurate software fault estimation method than SRGM.

2. Characteristics and Concerns of Software Reliability Growth Models (SRGM)

Software Reliability Growth Models (SRGM) are statistical approaches based on Non-Homogeneous Poisson Processes (NHPP).[8] These models are used to predict the number of initial faults and future fault reductions by modeling software fault reduction patterns over time.[3] Notable SRGM models include Goel-Okumoto, Jelinski-Moranda, Weibull-Type Testing Function, and Inflection S-Shaped models, each suitable for explaining specific fault discovery patterns.[6]

However, SRGM has several limitations due to its characteristic of estimating parameters and making predictions based on historical fault data. Particularly when the amount of data is insufficient or of low quality, prediction accuracy decreases. These limitations are more pronounced in the early stages of software testing or in situations with incomplete data.[3]

3. LSTM's Structural Mechanism and Time Series Prediction Effect

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) specifically designed to address the long-term dependency problem. While RNNs are effective in learning and predicting patterns in time series data, they have limitations in learning performance as time extends.

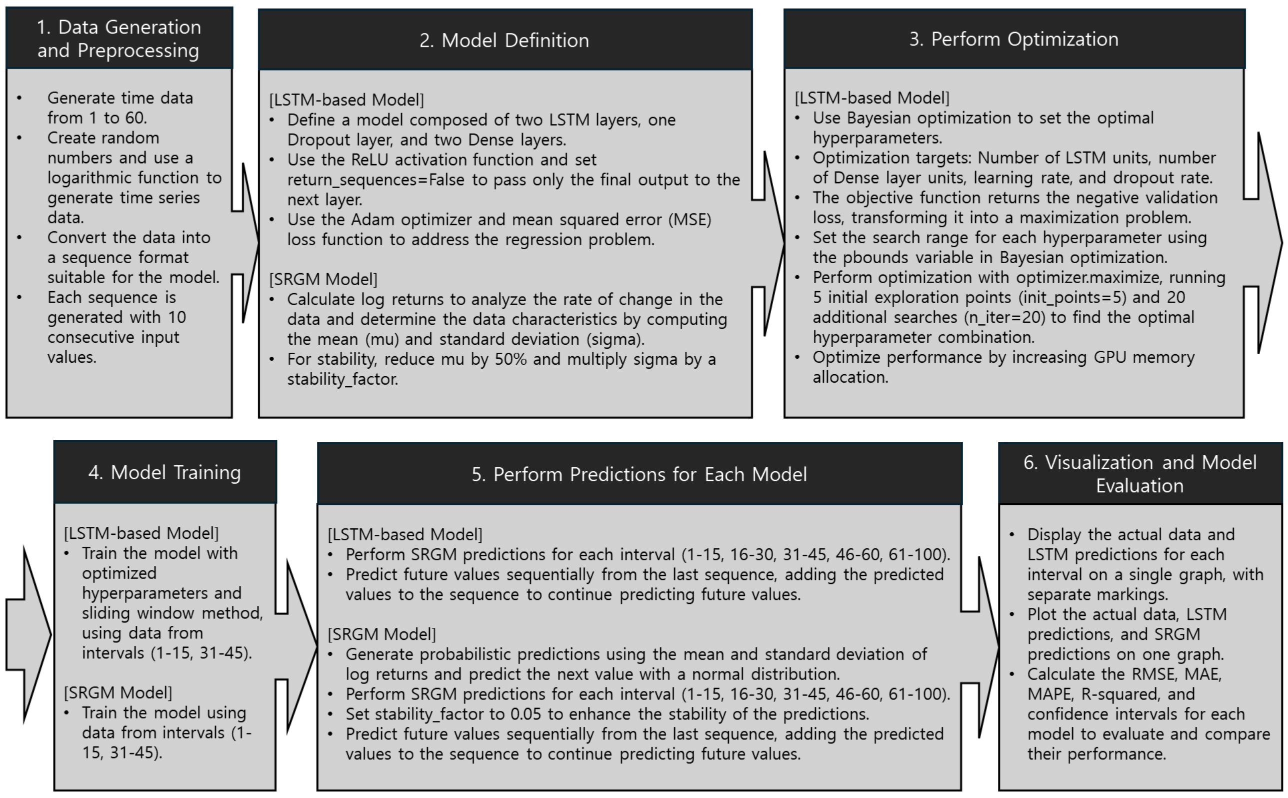
LSTM introduces 'memory cells' and 'gate' structures to solve this problem. Through these structures, LSTM can maintain complicated and important information for long periods and remove unnecessary information. This characteristic allows LSTM to effectively learn long-term time series patterns in data.[5]

These features of LSTM are particularly suitable for predicting software fault occurrences. It can learn and predict fault occurrence patterns at various time points, making it useful for software reliability analysis.

4. Efficiency of Hyperparameter Tuning through Bayesian Optimization

Bayesian optimization is a technique for hyperparameter tuning that is effective in searching for the global minimum. This method efficiently explores the parameter space to find the optimal combination of hyperparameters. Bayesian optimization adjusts hyperparameters to minimize validation loss, ensuring consistent performance across various initial conditions without falling into local minimum. Due to these characteristics, Bayesian optimization plays a crucial role in improving the performance of predictive models.

5. Research Method for Improving Software Reliability Prediction Using LSTM-Based Models



1. Data Generation and Preprocessing

- Generate time data from 1 to 60.
- Create random numbers and use a logarithmic function to generate time series data.
- Convert the data into a sequence format suitable for the model.
- Each sequence is generated with 10 consecutive input values.

2. Model Definition

[LSTM-based Model]
- Define a model composed of two LSTM layers, one Dropout layer, and two Dense layers.
- Use the ReLU activation function and set return_sequences=False to pass only the final output to the next layer.
- Use the Adam optimizer and mean squared error (MSE) loss function to address the regression problem.

[SRGM Model]
- Calculate log returns to analyze the rate of change in the data and determine the data characteristics by computing the mean (mu) and standard deviation (sigma).
- For stability, reduce mu by 50% and multiply sigma by a stability_factor.

3. Perform Optimization

[LSTM-based Model]
- Use Bayesian optimization to set the optimal hyperparameters.
- Optimization targets: Number of LSTM units, number of Dense layer units, learning rate, and dropout rate.
- The objective function returns the negative validation loss, transforming it into a maximization problem.
- Set the search range for each hyperparameter using the pbounds variable in Bayesian optimization.
- Perform optimization with optimizer.maximize, running 5 initial exploration points (init_points=5) and 20 additional searches (n_iter=20) to find the optimal hyperparameter combination.
- Optimize performance by increasing GPU memory allocation.

4. Model Training

[LSTM-based Model]
- Train the model with optimized hyperparameters and sliding window method, using data from intervals (1-15, 31-45).

[SRGM Model]
- Train the model using data from intervals (1-15, 31-45).

5. Perform Predictions for Each Model

[LSTM-based Model]
- Perform SRGM predictions for each interval (1-15, 16-30, 31-45, 46-60, 61-100).
- Predict future values sequentially from the last sequence, adding the predicted values to the sequence to continue predicting future values.

[SRGM Model]
- Generate probabilistic predictions using the mean and standard deviation of log returns and predict the next value with a normal distribution.
- Perform SRGM predictions for each interval (1-15, 16-30, 31-45, 46-60, 61-100).
- Set stability_factor to 0.05 to enhance the stability of the predictions.
- Predict future values sequentially from the last sequence, adding the predicted values to the sequence to continue predicting future values.

6. Visualization and Model Evaluation

- Display the actual data and LSTM predictions for each interval on a single graph, with separate markings.
- Plot the actual data, LSTM predictions, and SRGM predictions on one graph.
- Calculate the RMSE, MAE, MAPE, R-squared, and confidence intervals for each model to evaluate and compare their performance.

Figure 1. Research Method Model.

This study focuses on fault prediction in the software testing phase, designing and implementing an LSTM-based deep learning model to overcome the concerns of existing SRGM models. The aim is to propose a more accurate and efficient methodology for software fault prediction. As shown in **Figure 1**, the research process consists of data generation and preprocessing, model definition, optimization, model training, prediction, and visualization and model evaluation. LSTM-based models and SRGM models were designed separately, with predictions made and evaluated for each interval time. This comprehensive approach sought to improve the accuracy and reliability of fault prediction in the software testing phase.

5.1. Synthetic Dataset Construction

In this study, synthetic fault data was utilized for the software testing phase. Fault data for 1-60 hours of the software testing phase was generated, and time series data was created using random numbers and logarithmic functions. When converting data into sequence form suitable for the model, each sequence was generated consisting of 10 consecutive input values.

5.2. LSTM-Based Deep Learning Model Design and Learning Process

The LSTM model consists of an input layer, two LSTM layers, a dropout layer, two Dense layers, and an output layer. This structure was designed to effectively capture and learn complex patterns in time series data.

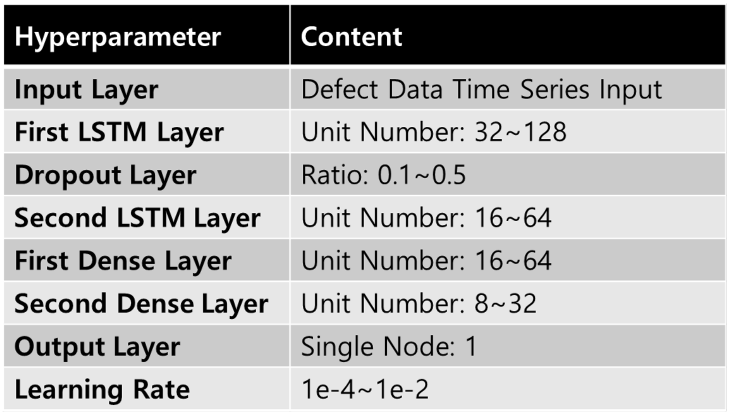
The main hyperparameters of the LSTM-based model were dynamically determined through Bayesian Optimization. The optimized hyperparameters in this process are as follows.

- Number of units in the first and second LSTM layers
- Number of units in the two Dense layers
- Learning rate
- Dropout rate

ReLU (Rectified Linear Unit) was adopted as the activation function to provide non-linearity to the model. A linear activation function was applied to the output layer to accurately predict the final fault occurrence.

The Bayesian Optimization process was performed as follows.

- Objective function definition: A function returning the negative value of validation loss was defined. This was designed to search for the minimum validation loss by maximizing this function.
- Search space setting: As shown in **Figure 2**, initial search ranges were set for each hyperparameter. This is to comprehensively consider factors that significantly affect model performance.
- Initial search and iteration: Starting from 5 initial points, the optimal hyperparameter combination was searched through a total of 20 iterations. This iterative process plays an important role in efficiently investigating the search space and finding the optimal solution.



Hyperparameter	Content
Input Layer	Defect Data Time Series Input
First LSTM Layer	Unit Number: 32~128
Dropout Layer	Ratio: 0.1~0.5
Second LSTM Layer	Unit Number: 16~64
First Dense Layer	Unit Number: 16~64
Second Dense Layer	Unit Number: 8~32
Output Layer	Single Node: 1
Learning Rate	1e-4~1e-2

Figure 2. Hyperparameter Structure and Initial Search Range Setting

This Bayesian Optimization approach allows for effective search of the global minimum without falling into local minimum. This contributes to improving the model's generalization performance and ensuring consistent performance under various initial conditions. Consequently, this methodology can greatly enhance the robustness and reliability of software fault prediction models.

Additionally, GPU memory increase settings were applied to optimize model performance. This allowed for larger batch sizes and more complex model structures, improving learning efficiency and model performance.

Furthermore, a sliding window technique was introduced to address the problem of data scarcity and secure necessary data. This technique divides data into specific intervals for learning, allowing for the generation of sufficient learning samples even from limited datasets. This improved the learning stability of the model and enabled more accurate predictions.

As a result, this comprehensive approach significantly improved the robustness and reliability of the software fault prediction model. The application of GPU optimization and sliding window techniques contributed to enhancing the learning efficiency of the model and maximizing data utilization, thereby improving prediction accuracy.

5.3. SRGM Model Implementation

The SRGM model predicts future values based on the logarithmic return rate of the data. The logarithmic return rate reflects the rate of change between data points and is used to perform predictions for future time points. In this study, the logarithmic return rate r_t was calculated and implemented as shown in (1).

$$r_t = \ln\left(\frac{y_{t+1}}{y_t}\right) \tag{1}$$

Here, y_t represents the cumulative number of faults at time t.

The main parameters of the SRGM model are the mean return rate (μ) and standard deviation (σ).

- Mean return rate (μ): To prevent sudden increases or decreases, the mean value of the logarithmic return rate is multiplied by 0.5 and calculated as shown in (2).

$$\mu = \mathbb{E}\,[r_t] \tag{2}$$

- Standard deviation (σ): The standard deviation of the logarithmic return rate is adjusted by multiplying it by a stability factor. This is to dynamically adjust σ according to the sample size of the data, and is expressed as shown in (3).

$$\sigma = \sqrt{\mathbb{E}\,[(r_t - \mu)^2]} \tag{3}$$

Through these parameters, the SRGM model models software fault patterns and predicts future fault occurrences.

5.4. Prediction and Performance Evaluation

In this study, an LSTM-based deep learning model was used to predict software fault occurrences at future time points. For prediction performance evaluation and model validation, the analysis was performed by dividing into five intervals as follows:

- Time points 1-15 and 31-45: The distribution of actual data and the distribution of predicted values from the trained model were visually compared and verified. This was used to evaluate the learning performance and suitability of the model.

- Time points 16-30 and 46-60: To verify the prediction accuracy for data not learned by the model, predicted values and actual data were visually compared and analyzed. This serves as an important indicator for evaluating the model's generalization ability.
- Time points after 61: The model's prediction ability was evaluated for future time points where actual data is not available. The long-term prediction performance of the model was analyzed by visually confirming whether it accurately reflects the fault occurrence trend from time points 1-60.

Through this segmented approach, we aim to comprehensively evaluate the model's learning performance, generalization ability, and long-term prediction capability. The specific prediction methods for each interval are as follows:

- Time points 1-15: Direct prediction using the model trained on actual data
- Time points 16-30: Continuous prediction using only learning data from time points 1-15, compared with actual data distribution
- Time points 31-45: Prediction by training a new LSTM model based on actual data
- Time points 46-60: Continuous prediction using only learning data from time points 31-45, compared with actual data distribution
- Time points after 61: Future prediction beyond the range of learning data using hyperparameters optimized from time points 1-60. This interval is an area without actual fault data, allowing for evaluation of the model's pure prediction ability. Due to graphing limitations, only up to time point 100 was displayed.

Through this multifaceted analysis, we can comprehensively evaluate the software fault prediction performance of the LSTM-based deep learning model and clearly identify the model's strengths and limitations.

Figure 3 below is a graph visualizing the actual data and predictions of the LSTM-based deep learning model divided into 5 intervals from time points 1 to 100.

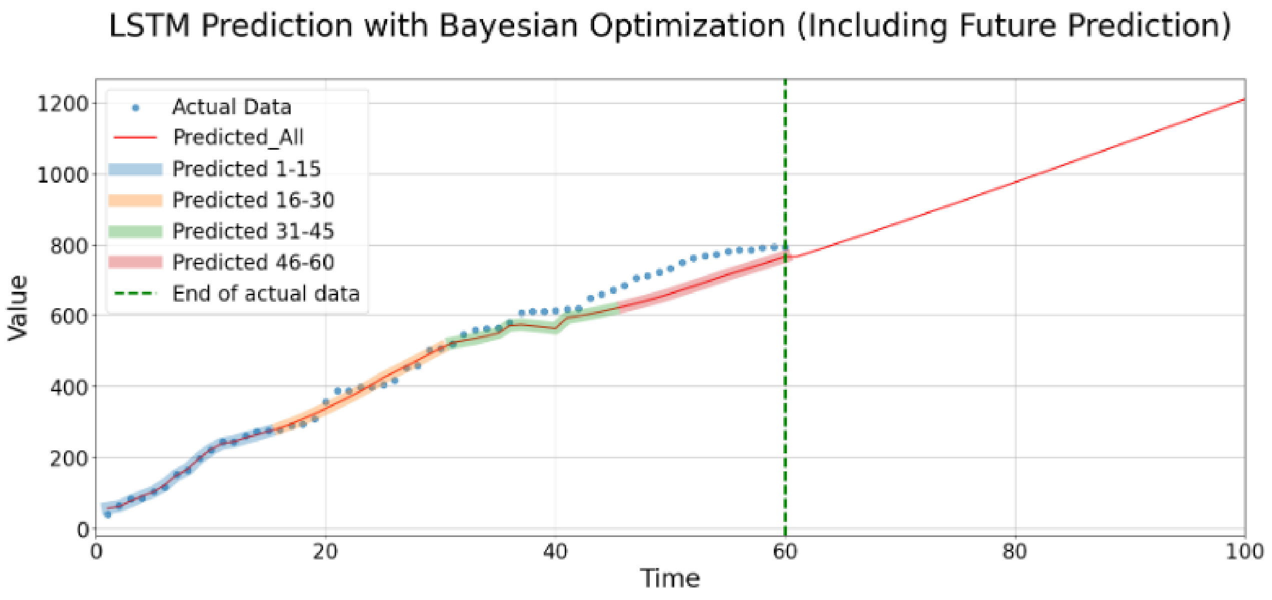
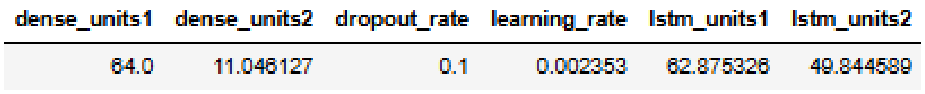


Figure 3. LSTM-based Deep Learning Model Prediction Graph for Each Interval

Figure 4 shows the optimal hyperparameter values applied when predicting with the LSTM-based deep learning model.



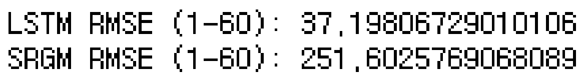
dense_units1	dense_units2	dropout_rate	learning_rate	lstm_units1	lstm_units2
64.0	11.046127	0.1	0.002353	62.875326	49.844589

Figure 4. Final Hyperparameter Search Results

The prediction performance was evaluated using RMSE (Root Mean Square Error).[9] RMSE is the square root of the average of the squared differences between predicted and actual values, with lower values indicating better prediction performance. The formula is shown in (4).

$$RMSE(\theta_1,\theta_2) \quad = \sqrt{MSE(\theta_1,\theta_2)} = \sqrt{E((\theta_1 - \theta_2)^2)} = \sqrt{\frac{\sum_{i=1}^{n}(x_{1,i}-x_{2,i})^2}{n}} \tag{4}$$

The prediction results were further analyzed by comparing with SRGM, confirming that the LSTM-based deep learning model showed superior prediction performance compared to SRGM. The LSTM-based deep learning model applying Bayesian optimization technique improved prediction accuracy by searching for the global minimum, and showed high performance even in situations with insufficient data compared to the SRGM model. As shown in **Figure 5**, the RMSE value in the 1-60 interval differed by more than 200.



```
LSTM RMSE (1-60): 37.19806729010106
SRGM RMSE (1-60): 251.6025769068089
```

Figure 5. RMSE Performance Comparison Metrics of LSTM-based Deep Learning Model and SRGM Model

Figure 6 visually shows the overall prediction pattern of the model and the difference from the actual data, providing an intuitive comparison.

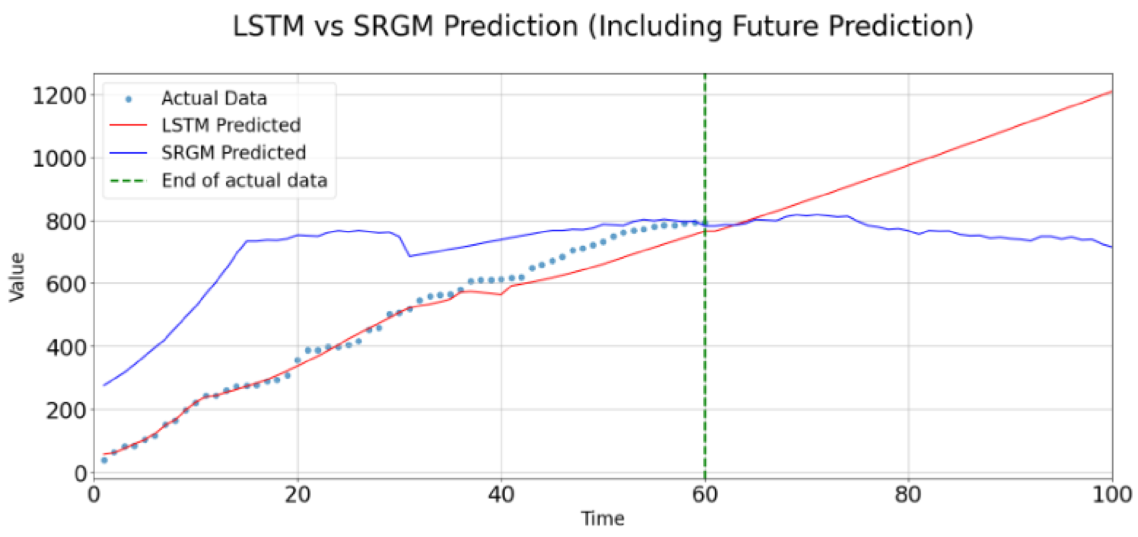


Figure 6. Performance Comparison Graph of LSTM-based Deep Learning Model and SRGM Model.

Predictions for time points after 61 were performed to evaluate the model's performance in situations where actual data is not available. The LSTM-based deep learning model produced stable prediction results by accurately reflecting the data trend even in this interval.

In this study, we effectively utilized the characteristics of time series data by applying the sliding window technique, and aimed to achieve high performance even with limited test data through hyperparameter tuning and model structure improvement via Bayesian optimization and through GPU memory increase settings.

6. Research Results

The LSTM-based deep learning model showed high prediction accuracy compared to the existing SRGM model even in situations with insufficient fault data in the software testing phase. Specifically, the LSTM-based deep learning model showed superior performance with an RMSE more than 200 lower than the SRGM model in the 1-60 time point interval. The RMSE of the LSTM model was about 37, while the RMSE of the SRGM model was about 251, showing that the LSTM model had an error 214 lower. This means that the LSTM-based deep learning model has a smaller difference between the actual fault data and the prediction results compared to the SRGM model.

The interval analysis results showed the following performance:

- Time points 1-15 and 31-45: The LSTM model predicted the actual data distribution very accurately.
- Time points 16-30 and 46-60: It showed high prediction accuracy even for data not used in training.
- After time point 61: Despite the absence of actual data, the LSTM-based deep learning model performed predictions that well reflected the trend of existing fault data.

These results indicate that the LSTM-based deep learning model has strengths in predicting future fault occurrences by effectively learning long-term patterns in time series data. It also suggests that the LSTM-based deep learning model can effectively complement the limitations of SRGM, which has difficulty in making predictions when data is scarce.

As a result of applying the Bayesian optimization technique, the LSTM-based deep learning model was able to efficiently search for the global minimum, avoiding local minimum and producing more stable prediction results. The optimized hyperparameters shown in **Figure 4** maximized the model's performance and minimized validation loss.

The LSTM-based deep learning model incorporating the sliding window technique and GPU memory increase settings showed more accurate prediction performance even in data-scarce situations compared to the SRGM model. As can be seen in **Figure 6**, the prediction curve of the LSTM model follows the trend of the actual data more closely. This demonstrates that while the SRGM model's performance degrades in data-scarce situations, the LSTM model can effectively learn and predict even with limited data.

In conclusion, this study confirmed that the proposed LSTM-based deep learning model overcomes the concerns of existing SRGM models in software fault prediction and provides more accurate and stable prediction performance.

7. Conclusion

This study proposes an LSTM-based deep learning model for fault prediction in the software testing phase and compares and analyzes its performance with the existing SRGM model. The research results confirmed that the LSTM-based model demonstrated superior prediction accuracy compared to traditional SRGM, even in situations with limited fault data.

Key findings of this study are as follows.

- Prediction Accuracy: The LSTM model achieved higher prediction accuracy, demonstrating an RMSE 214 lower than the SRGM model in the 1-60 time interval.
- Response to Data Scarcity: Even in environments with limited data, the LSTM model exhibited more stable and accurate predictive performance compared to the SRGM model.
- Long-term Prediction Capability: The LSTM model effectively predicted trends reflecting existing fault data, even in intervals beyond the 61st time point where actual data was unavailable.
- Model Optimization: Model performance was maximized through hyperparameter tuning using Bayesian optimization techniques.
- Data Utilization Efficiency: The application of the sliding window technique and GPU memory increase enabled effective learning even with limited datasets.

These results suggest that the LSTM-based deep learning model has significant advantages in learning long-term and complicated patterns of time series data and predicting future fault occurrences in software reliability analysis. The model effectively complements the concerns of SRGM, particularly in situations where data is scarce or incomplete.

These results suggest that LSTM-based deep learning models can overcome the concerns of existing SRGM models in the field of software fault prediction, providing more accurate and stable predictions. The LSTM model is particularly effective in situations with limited data or when long-term predictions are required.

Future research could further generalize these findings by validating the results using data from various real software projects and conducting comparative analyses with other deep learning models. Additionally, research into methods for improving model interpretability and providing reliability measures for prediction results is necessary.

In conclusion, the LSTM-based deep learning model proposed in this study presents new possibilities in the field of software reliability prediction. It is expected to be particularly useful in areas requiring high reliability, such as the defense industry.

Acknowledgement

This work was supported by the Technology Spin-off Project conducted by the Institute of Civil-Military Technology Cooperation of Agency for Defense Development, with funding from the Government of the Republic of Korea (Ministry of Trade, Industry and Energy and Defense Acquisition Program Administration) since 2023 (Development of AI-based Software Reliability Analysis and Management Program (23-SF-AI-05), November 2023 - October 2025)

References

[15] IEEE Recommended Practice on Software Reliability. (2017). IEEE Std 1633-2016 (Revision of IEEE Std 1633-2008) (pp.1-261)

[16] Lyu, M. R. (1996). Handbook of Software Reliability Engineering. IEEE Computer Society Press.

[17] K. K. San, H. Washizaki, Y. Fukazawa, K. Honda, M. Taga and A. Matsuzaki. (2019). DC-SRGM: Deep Cross-Project Software Reliability Growth Model. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 61-66).

[18] Pradhan V, Patra A, Jain A, Jain G, Kumar A, et al. (2024). PERMMA: Enhancing parameter estimation of software reliability growth models: A comparative analysis of metaheuristic optimization algorithms. PLOS ONE 19(9)

[19] F. Yangzhen, Z. Hong, Z. Chenchen and F. Chao. (2017). A Software Reliability Prediction Model: Using Improved Long Short Term Memory Network. IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 614-615)

[20] Youn Su Kim, Hoang Pham, and In Hong Chang. (2023). Deep-Learning Software Reliability Model Using SRGM as Activation Function. Applied Science, 13(19).

[21] N. Karunanithi, D. Whitley and Y. K. Malaiya. (1992). Using neural networks in reliability prediction. IEEE Software, 9(4), 53-59

[22] 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), 206-211.

[23] Kumar, P., Singh, Y. (2012). An empirical study of software reliability prediction using machine learning techniques. International Journal of System Assurance Engineering and Management, 3, 194–208.