

Exploring the recurrent and sequential security patch data using deep learning approaches

Falah Muhammad Alam, Devi Fitriana

Department of Technology Information, Faculty of Computer Science, Binus University, Jakarta, Indonesia

Article Info

Article history:

Received Aug 18, 2024

Revised Apr 10, 2025

Accepted May 24, 2025

Keywords:

Bidirectional long short-term memory

Deep learning

Gated recurrent unit

Long short-term memory

Recurrent neural networks

Security patches

ABSTRACT

The ever-changing nature of vulnerabilities and the intricacy of temporal connections make the classification of security patch data, both sequential and recurrent, a formidable challenge in cybersecurity. The goal of this research is to improve the efficacy and precision of security patch management by optimizing deep learning models to deal with these issues. In order to assess their performance on the PatchDB dataset, four models were used: recurrent neural networks (RNN), long short-term memory (LSTM), gated recurrent unit (GRU), and bidirectional LSTM (Bi-LSTM). Metrics like F1-score, area under the receiver operating characteristic curve (AUC-ROC), recall, accuracy, and precision were used to evaluate performance. When it came to processing sequential data, the GRU model was the most efficient, with the best accuracy (77.39%), recall (65.63%), and AUC-ROC score (0.8127). With a 75.17% accuracy rate and an AUC-ROC score of 0.7752, the RNN model successfully reduced false negatives. With AUC-ROC scores of 0.7792 and 0.8055, respectively, LSTM and Bi-LSTM had better specificity but more false negatives. To improve cybersecurity operations, decrease mitigation time, and automate the classification of security updates, this study presents a methodology. To improve the models' practicality, future efforts will center on increasing datasets and testing them in real-world settings.

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



Corresponding Author:

Falah Muhammad Alam

Department of Technology Information, Faculty of Computer Science, Binus University

Kebon Jeruk St. No.27, West Jakarta 11530, DKI Jakarta, Indonesia

Email: falah.alam@binus.ac.id

1. INTRODUCTION

The fast development of digital technology and the rising dependence on software infrastructure in many sectors have made cybersecurity a paramount concern in the recent decade. By fixing previously discovered software vulnerabilities, security patches protect systems from a broad variety of new threats [1]. The dangers of cyberattacks which can result in substantial financial losses, reputational harm, and interruptions to operations must be mitigated by applying these fixes in a timely and effective manner. But there are a lot of obstacles to overcome when analyzing and deploying security patches. This is especially true because patch data is sequential and recurring, which makes analysis and prediction more difficult [2].

It is necessary to apply updates in a precise sequence to fix vulnerabilities as they occur, and this sequential pattern is typically seen in security patch data. The fact that updates are often applied to multiple software versions to address the same or similar vulnerabilities further complicates matters [3]. Static analysis and rule-based approaches are examples of traditional methods that use heuristics and predefined rules to find vulnerabilities. Although these methods work well in static situations, they can't handle data that is intrinsically sequential and time-sensitive, which makes it difficult to capture how software vulnerabilities

change over time [4]. This shortcoming has prompted research into more sophisticated methods that can handle the intricacies of security patch data.

Because of its capacity to handle massive amounts of complicated data and discover patterns that conventional approaches frequently fail to notice, deep learning has arisen as a potential option for evaluating sequential and temporal data [5]. Intrusion detection, malware categorization, and vulnerability assessment are just a few of the cybersecurity applications where deep learning models have been shown to be effective in recent studies [6]. Security patch classification tasks are well-suited for models like recurrent neural networks (RNN), long short-term memory (LSTM), gated recurrent units (GRU), and bidirectional LSTM (Bi-LSTM) because of their effectiveness in learning long-term dependencies within sequences [7], [8]. When it comes to these, Bi-LSTM stands out since it takes into account both the past and the future, which improves its prediction abilities in situations where the sequence of events is crucial [7].

Examining and contrasting the performance of RNN, LSTM, GRU, and Bi-LSTM models in dealing with sequential and recurrent security patch data is the main goal of this research. This work thoroughly evaluates these models using the PatchDB dataset. It employs rigorous experimental approaches such as data preparation, hyperparameter tuning, training, validation, and testing. To find the best method for security patch classification, scientists employ metrics including recall, accuracy, precision, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) to evaluate model performance [9]–[11].

This study does more than just compare model performance; it also looks into how hyperparameter adjustment can improve deep learning models' efficacy [12]. This project seeks to develop a strong framework for analyzing security patch data by determining optimal configurations. Because they provide useful information about how to use deep learning to automate patch management processes, the results should have a major impact on cybersecurity. In the end, this research sets the stage for creating sophisticated systems that enhance the dependability and effectiveness of cybersecurity processes, decreasing the need for human intervention and lowering the risks linked to software vulnerabilities.

2. THE COMPREHENSIVE THEORETICAL BASIS

Grasping how deep learning architectures address the sequential and repetitive elements intrinsic to security patch data is paramount for ensuring accurate patch classification. RNN, LSTM, GRU, and Bi-LSTM are specifically designed to effectively capture temporal dependencies in sequential datasets. By leveraging their distinctive ability to retain historical context and manage data sequences with varying complexities, these architectures significantly improve the classification accuracy of security patches.

2.1. Recurrent and sequential data in cybersecurity

Patches are typically issued in a sequential fashion to address different software vulnerabilities as they are found, because security patch data is naturally recurrent and sequential. Because of this sequential character, the data takes on a time dimension; the sequence and timing of patch releases can have a major effect on a system's overall security. These temporal dependencies are crucial for good vulnerability management, yet traditional approaches to vulnerability detection, such as static analysis methods, generally fail to capture them [1]. Research has demonstrated that static analysis methods have their uses, but they frequently overlook software vulnerabilities' ever-changing nature, especially when they happen repeatedly and sequentially [5]. RNNs and LSTM networks, which are specifically designed for sequential data, provide a potential solution to these problems by accurately modelling the temporal relationships in the data, which improves the reliability and accuracy of vulnerability detection [2].

2.2. Recurrent neural networks (RNN)

Time series analysis is one area where the sequence of data points is very important, and one of the fundamental designs for handling sequential data is the RNN [8]. RNNs learn dependencies over time by preserving information from past time steps in a hidden state. The vanishing gradient problem is a well-known issue with RNNs; it happens when the gradients utilized in backpropagation get too small, preventing the network from learning data dependencies over the long term. When working with lengthy sequences, this issue can severely impair RNN performance, and it is especially noticeable in deep networks [13]. In spite of these obstacles, RNNs have established a foundation for comprehending sequential input, and their design has prompted more sophisticated models such as LSTM networks, which overcome some of these shortcomings by incorporating techniques to maintain gradients throughout time [8].

2.3. Long short-term memory networks (LSTM)

To overcome the issues with regular RNNs, such as the vanishing gradient problem that prevents RNNs from learning long-term dependencies in sequential data, LSTM networks were created [8]. LSTMs get around this problem by incorporating memory cells and gating mechanisms that let the network keep and

update data for long periods of time, thereby capturing long-term dependencies [14]. Natural language processing, time series forecasting, and the analysis of sequential data (such as security patches) are some examples of jobs that LSTMs excel at because of these capabilities. With its ability to simulate the temporal connections between software updates and improve the speed and accuracy of vulnerability identification through the study of system logs, LSTMs have demonstrated great promise in the domain of cybersecurity. They also improve the prediction of the effectiveness of security patches. Because of these features, LSTMs are an effective tool for protecting software systems from new threats, especially in situations where applying security fixes accurately and in a timely manner is critical.

2.4. Gated recurrent units (GRU)

To simplify LSTM network architecture while keeping their efficiency in capturing temporal dependencies, a relatively recent development in recurrent neural network designs are GRUs [15]. In order to reduce computational complexity without sacrificing performance, GRUs merge the input and forget gates into one single gate. For situations with constrained computing resources, this makes GRUs superior to LSTMs [16]. Because of their advantageous trade-off between computational complexity and performance, GRUs have found useful use in real-time vulnerability detection systems within the cybersecurity domain [6]. Because of these features, GRUs are ideal for real-time cybersecurity applications because of the solid performance they provide while processing sequential input quickly and accurately.

2.5. Bidirectional LSTM (Bi-LSTM)

By combining the best features of forward and backward processing, bidirectional LSTM networks improve upon traditional LSTMs and enable models to detect dependencies that would otherwise go unnoticed. With this two-way approach, Bi-LSTMs can think about the past and the future at the same time, which helps them understand the data's temporal linkages better. Using its capacity to analyze sequences more deeply, Bi-LSTMs have been successfully used to improve the detection of vulnerabilities in security patches. This has led to deeper insight into potential security threats and better prediction accuracy in cybersecurity applications [7].

2.6. Summary and application of deep learning in vulnerability detection

A number of recent studies have concentrated on the use of RNN, LSTM, GRU, and Bi-LSTM deep learning models in cybersecurity, namely in the areas of vulnerability tracking and patch administration. Security patch data presents unique issues, but each model has its own benefits when it comes to handling sequential data. Researchers hope to strengthen software systems' security by making better use of these models to discover vulnerabilities more quickly and accurately.

3. METHOD

3.1. Overview of methodology

In order to classify security patches effectively, this study's technique takes into account the PatchDB dataset's recurrent and sequential features. This section gives a detailed explanation of the dataset, including its statistics and its unique features including the deployment of patches sequentially throughout time. In order to deal with the time-sensitive data, we offer comprehensive preparation procedures, such as tokenization and normalization. By visualizing and uncovering distributions and patterns within the dataset across time, exploratory data analysis (EDA) provides useful insights for developing models. Additionally, the technique provides an explanation for why specific deep learning models were chosen, including RNN, LSTM, GRU, and Bi-LSTM, all of which excel at capturing dependencies in sequential datasets. We also go over the assessment measures used, including accuracy, precision, recall, and AUC-ROC, with an emphasis on how they pertain to sequential data processing, and how to optimize the model's performance through hyperparameter tuning and advanced training procedures. The difficulties of security patch categorization can be properly addressed with the help of this organized method.

3.2. Research stages

As presented in Figure 1, the proposed study employs a structured, multi-phased methodology to classify security patches using deep learning models. The process begins with data acquisition and preliminary analysis, which aim to explore the dataset's characteristics thoroughly. Next, essential preprocessing steps are performed: commit messages are combined with the associated code differences, and text data are transformed using term frequency-inverse document frequency (TF-IDF) tokenization. Once the textual data has been prepared, the dataset is divided into training and testing subsets to ensure a reliable assessment of model performance.

Following data preparation, various deep learning architectures namely RNN, LSTM, GRU, and Bi-LSTM are developed and fine-tuned through hyperparameter optimization to achieve superior accuracy. After identifying the optimal configurations, each model is trained and evaluated using appropriate metrics to gauge its predictive capabilities. The evaluation phase highlights the strengths and weaknesses of every model, shedding light on their respective effectiveness in handling sequential and repetitive patch data. This comprehensive approach ultimately enhances software security management by enabling swift and precise detection of security patches.

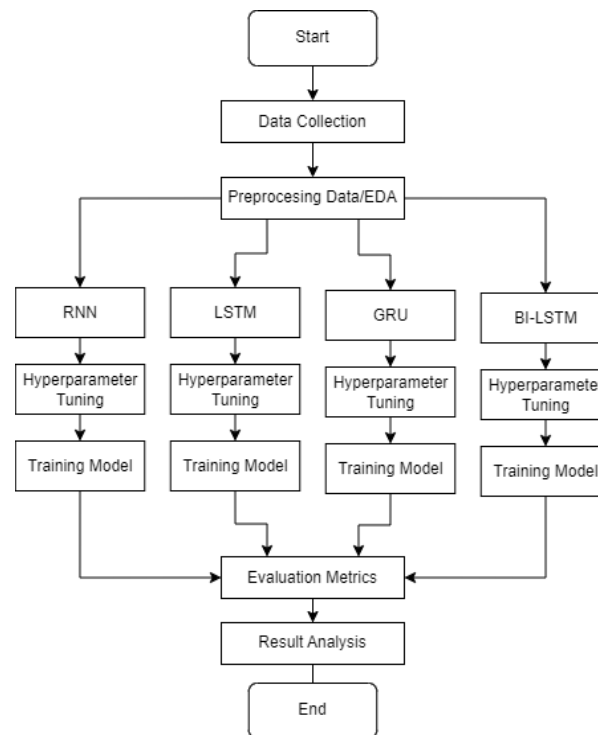


Figure 1. The proposed multi-stage research framework for security patch classification

3.3. Data collection

In this part, the dataset that was utilized for training and assessing deep learning models for sequential security patch categorization is described, with an emphasis on its importance. The SunLab-created "PatchDB: A large-scale security patch dataset" [17], contains both security and non-security patches culled from well-known GitHub projects and the National Vulnerability Database (NVD). A perfect fit for sequential and temporal data analysis, it collects commit and diff attributes together with multi-labels ("Security" and "Non-security"). To start gathering data, we crawled the NVD and common vulnerabilities and exposures (CVE) databases for documented security patches, making sure to include only validated patches. By adding real-world patches to GitHub commits and using oversampling techniques to balance the quantity of security and non-security changes, we were able to address class imbalance and promote diversity. A balanced, diversified, and high-quality dataset that is well-suited for deep learning applications was achieved using this all-encompassing methodology. In order to achieve reliable and precise security patch categorization, the study makes use of PatchDB and the capabilities of deep learning models including RNN, LSTM, GRU, and Bi-LSTM. These models excel at detecting patterns and dependencies in sequential data over the long term.

3.4. Preprocessing data

Data quality, consistency, and suitability for sequential deep learning models were guaranteed through a thorough sequence of preparation processes that were performed on the dataset to get it ready for modelling. This process began with exploratory data analysis (EDA) to identify distributions, trends, and potential anomalies in the patch dataset. The EDA included class imbalance inspection, token frequency distribution, and sequence length analysis to ensure compatibility with sequential models. Subsequently, data

cleaning and preparation steps such as handling missing values and formatting data structures were applied to ensure the dataset met the input requirements for time-series deep learning algorithms.

3.4.1. Exploratory data analysis (EDA)

The EDA of the dataset reveals a substantial class imbalance between security and non-security patches, which was addressed by oversampling during data preprocessing. Specifically, the number of non-security patches significantly outnumbered the security patches, which could lead to model bias if left unhandled. This imbalance can be seen clearly in Figure 2, where the bar chart highlights the differing counts between these two patch types.

Figure 2 also includes a pie chart that illustrates how 88.6% of the patches are considered wild, while 11.4% come from the common vulnerabilities and exposures database. These visual representations not only showcase the dataset's diversity but also underscore the importance of balancing strategies for improved classification performance. By recognizing and correcting the dominance of wild patches at this early stage, the subsequent steps of data preparation and deep learning model development were better positioned to achieve robust and accurate security patch classification.

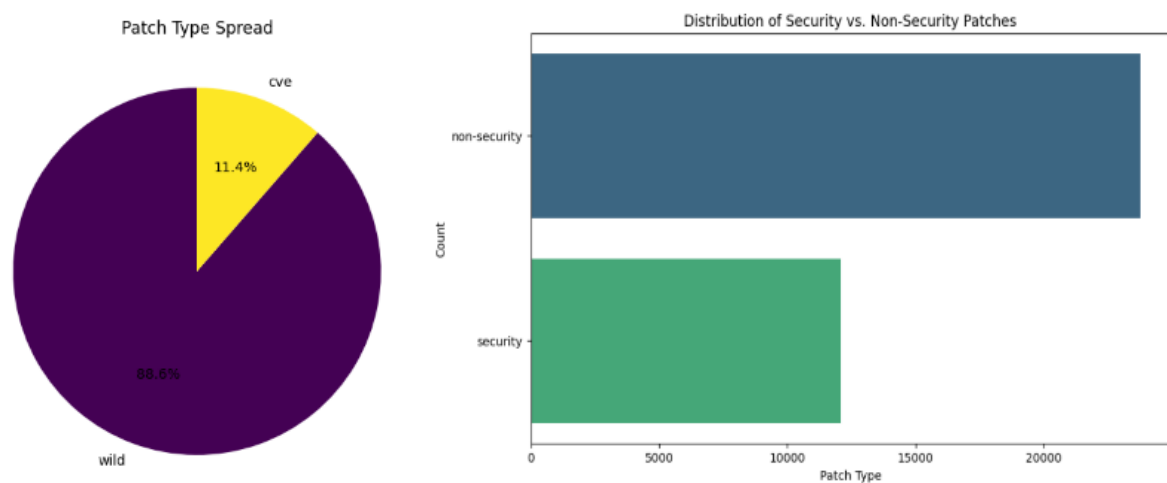


Figure 2. Part of the EDA, with a pie chart showing the patch type spread and a bar chart comparing security and non-security patches

3.4.2. Data cleaning

A thorough data cleaning process was carried out to preserve the dataset's consistency, reliability, and integrity, making it appropriate for deep learning models. To eliminate unnecessary repetition, we eliminated duplicate entries and combined text fields like commit messages and code diffs into one standard style [4]. up order to fill up any gaps or missing data, we checked with other resources, such as the NVD and GitHub [17]. A high-quality dataset that reduced biases and enhanced the models' reliability during training and evaluation was produced by meticulously reviewing each item to guarantee uniqueness and completeness.

3.4.3. Label encoding

It was critical to transform the categorical labels into a numerical format that the algorithms could interpret successfully in order to prepare the dataset for deep learning models. "Security" and "Non-security" are examples of textual labels that were converted into numerical values (e.g., 1 and 0) by label encoding [4]. Because of this change, the models were able to train appropriately on the categories, which allowed them to discover useful links and patterns in the data. Achieving reliable predictions and a smooth dataset integration with deep learning frameworks relied heavily on the label encoding process.

3.4.4. Tokenization and vectorization

The transformation of raw text into numerical representations was crucial for the deep learning models to efficiently process textual data. Term frequency-inverse document frequency (TF-IDF) vectorization and tokenization allowed us to achieve this. First, the combined text fields, including code diffs

and commit messages, were tokenized to make them easier to work with [4]. To make them usable as input to the model, these tokens were converted into numerical vectors with a predetermined amount of features. Tokenization and vectorization preserved the relevance of terms based on their frequency and uniqueness within the dataset, improving the text data format for deep learning applications. This allowed the models to train and predict well [18].

3.4.5. Data Splitting

Separating the dataset into subsets for training, validation, and testing was crucial to provide a strong and impartial assessment of the models. Stratified sampling was used to ensure that the "Security" and "Non-security" categories were evenly distributed throughout all subsets of the dataset, which was then divided into three parts: 70% for training, 15% for validation, and 15% for testing [3]. This method prevented overfitting during training while still providing the models with enough data for learning. A separate metric for the model's efficacy was supplied by the test set, which allowed for hyperparameter tuning and performance monitoring in the validation set. The evaluation process consistently evaluated the models' capacity to generalize to unknown data by utilizing this structured splitting strategy [12].

3.4.6. Define model (RNN, LSTM, GRU, BI-LSTM)

We developed and deployed four deep learning architectures RNN, LSTM, GRU, and Bi-LSTM to efficiently handle and categorize sequential security patch data. In order to capture the semantic links between words, each model design started with an embedding layer that turned input tokens into dense vectors [18]. Adapted to the specifics of each model, the following recurrent layers were built upon these embeddings. To handle long-term dependencies, the LSTM model made use of its gating mechanisms, whereas the RNN model learnt sequential dependencies using a SimpleRNN layer [8]. A GRU layer was used by the computationally efficient GRU model [15], while a Bidirectional LSTM layer was used by the bidirectionally efficient Bi-LSTM model [7] to process data in both directions. Probability scores for binary classification were generated using a dense output layer using a sigmoid activation function. For accurate predictions and efficient learning, all models were fine-tuned with the Adam optimizer and built with the binary cross-entropy loss function. The models were able to accurately classify security patch data because of the architectural design's emphasis on sequential and temporal patterns.

3.5. Hyperparameter tuning

To maximize the efficiency of a deep learning model, hyperparameter tuning is an essential process for determining the best possible parameter settings. Using an Optuna-based approach a state-of-the-art optimization framework, we methodically investigated several hyperparameter combinations in this study. Optuna finds the best settings by intelligently sampling setups, as opposed to grid search, which evaluates every potential parameter value exhaustively [12]. The study's critical hyperparameters were the following: learning rate, batch size, epochs, number of layers, number of units in recurrent layers, and number of layers overall. Model accuracy, recall, precision, F1-score, and AUC-ROC were all maximized using this procedure. Research has shown that hyperparameter tweaking has a major effect on the accuracy and computational efficiency of models, among other things [19], [20]. To determine the efficacy of each collection of hyperparameters, models were tested with a validation subset. Consistent and very accurate model classification of sequential security patch data was achieved as a consequence of this rigorous tuning procedure.

3.6. Training models

In deep learning applications with sequential data, proper model training is crucial for achieving optimal performance while minimizing overfitting. Here, we optimized RNN, LSTM, GRU, and Bi-LSTM models using state-of-the-art training methods. Because of its capacity to dynamically adjust learning rates and expedite convergence, the Adam optimizer is well-known for its efficiency in deep learning tasks, and it was used to train all of the models [21]. Throughout the training process, the model's performance was constantly tracked on a validation set using a defined number of epochs. Early stopping was used to end training when validation performance did not show any additional improvement; this was done to further improve generalizability and prevent overfitting [22]. By using this approach, we could be certain that our models would reliably detect important patterns in our training data and continue to perform admirably when presented with new, unknown test data [23]. Our models for categorizing security patch data have proven to be reliable and effective by using these training procedures.

3.7. Evaluation metrics

To properly evaluate deep learning models' success in identifying security fixes, it is crucial to choose appropriate assessment measures. This study thoroughly evaluated the efficacy of the model by using

a combination of generally recognized categorization measures. Precision measures the dependability of positive predictions, recall evaluates the ability to identify true positives, F1-score strikes a balance between recall and precision, and AUC-ROC measures the model's capacity to distinguish between classes across different thresholds [24]. Accuracy is used for general performance assessment. All of these measures worked together to provide a thorough assessment, showing how effectively each model dealt with sequential security patch data and where it fell short.

3.7.1. Confusion matrix

Classification model performance can be better understood with the help of tools that reveal their predictions in great detail. For example, the confusion matrix classifies forecasts as either true positive (TP), true negative (TN), false positive (FP), or false negative (FN). The model's predicted accuracy can be thoroughly evaluated thanks to this breakdown [9]. The confusion matrix is a useful tool for identifying the model's strengths and weaknesses, such as its positive and negative instance classification accuracy and the extent to which it misclassifies data [10]. An overview of the confusion matrix structure is provided in Table 1, which shows how the model's performance is evaluated by comparing predictions with actual outcomes.

Table 1. Confusion matrix

	Predicted positive	Predicted negative
Actual positive	True positive (TP)	False negative (FN)
Actual negative	False positive (FP)	True negative (TN)

3.7.2. Accuracy

A popular metric, accuracy shows how many instances out of the total number of instances in the dataset were properly predicted. An easy-to-understand overview of the model's performance is provided by it [16]. But it might be deceiving in datasets where there is a large disparity across classes [25]. Accuracy risks giving an exaggerated picture of the model's efficacy when one class is substantially more numerous than the other [26]. Equation (1) shows the formula for calculating accuracy, where TP and TN are the number of correctly categorized instances and total instances are the total data size:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}} \quad (1)$$

3.7.3. Precision

An important measure for assessing the accuracy of a model is its precision, which is defined as the percentage of correct predictions relative to the total number of correct predictions. In contexts where false positives have substantial implications, like medical diagnosis or fraud detection, a high precision represents a low false positive rate, which is especially crucial [27]. In unbalanced datasets, where the false positive cost could exceed the false negative cost, this statistic becomes even more important [10]. Preciseness guarantees meaningful and dependable model predictions, especially in high-stakes applications, by focusing on reducing the number of false positives. The formula for calculating precision is given by (2), which divides the total number of true positives (TP) by the sum of all true positives and false positives (FP):

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (2)$$

3.7.4. Recall

The sensitivity or recall of a model is defined as the percentage of correct predictions relative to the total number of correct predictions. If the model has a high recall, it means it successfully identifies a large percentage of true positives. This is especially important in medical screening and security applications where false positives can have serious implications [28]. In illness diagnosis, for instance, a high recall rate guarantees the detection of the majority of cases, notwithstanding the possibility of some false positives [11]. Recall, which shows how well the model identifies the minority class which is frequently more important is a crucial parameter to consider when working with extremely imbalanced datasets [26]. To find the recall, we divide the total number of true positives (TP) by the total number of true negatives (FN) using the formula given in (3):

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (3)$$

3.7.5. F1-score

An important metric for balancing recall and precision is the F1-score, which is the harmonic mean of the two. It offers a single metric that captures the accuracy of positive predictions (precision) and the completeness of the model's positive detections (recall) two important metrics in a dataset with an uneven class distribution [29]. In cases when one group is grossly under-represented, this statistic becomes crucial in preventing an overemphasis on either recall or precision [30]. For a thorough evaluation of the model's performance, the F1-score is widely employed in classification and information retrieval tasks to measure the trade-off between recall and precision. Using a combination of recall and precision, the F1-score is calculated according to (4):

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

3.7.6. Area under the ROC curve (AUC-ROC)

One important measure for assessing a model's class discrimination capabilities is the AUC-ROC. The ROC curve gives a complete picture of the model's performance across all classification thresholds by plotting the true positive rate (TPR) versus the false positive rate (FPR) at different threshold settings [9]. By reflecting the probability that the model rates a randomly selected positive instance higher than a randomly chosen negative one, a bigger area under the curve (AUC) signifies better performance, which is a measure of separability [31]. In cases where the dataset is not evenly distributed, this statistic becomes invaluable, as accuracy alone can distort the picture of how well a model is doing [32].

4. RESULTS AND DISCUSSION

Our study's findings break down each model's training and validation procedures in great detail. Along with hyperparameter adjustment and performance graphs, we also provide discussion and comparative analysis using evaluation measures. Furthermore, we present a case study that illustrates how the tested models were applied in practical situations and go over the results.

4.1. Hyperparameter tuning result

Table 2 shows the outcomes of hyperparameter tuning for our RNN, LSTM, GRU, and Bi-LSTM deep learning models assessed across several configurations. During the model creation process, the chosen hyperparameters, such as embedding size, number of units, and learning rate, are displayed in each row. These settings will be used to achieve optimal performance. The LSTM model attained a score of 0.6694 with an embedding size of 125, 246 units, and a learning rate of 0.00243600, in contrast to the RNN model's 0.6307, which was produced by an embedding size of 146, 74 units and a learning rate of 0.00015730. With 120 embeddings, 75 units, and a learning rate of 0.00282000, the GRU model achieved an impressive score of 0.6762, surpassing all of its competitors. With 289,254 units of embedding size and 0.00018620 units of learning rate, the Bi-LSTM model achieved a score of 0.6697. These results highlight the need of customizing the hyperparameter settings for every model to get the best outcomes. Hyperparameter combinations including embedding size and learning rate can dramatically affect the efficacy and precision of deep learning models, as demonstrated by the GRU model's outperformance. For tasks involving the classification of security patches in sequential and recurrent data, it is particularly important to tune and carefully implement these parameters during the model-building process.

Table 2. Hyperparameter tuning best result

Model	Trial	Embedding	Units	Learning rate	Value (Score)
RNN	6	146	74	0.00015730	0.6307
LSTM	2	125	246	0.00243600	0.6694
GRU	15	120	75	0.00282000	0.6762
Bi-LSTM	8	289	254	0.00018620	0.6697

4.2. Evaluation metrics

Table 3 demonstrates that while other models have lower false negative rates, the RNN model (Default) has a more even performance. With the help of adjustment, the RNN model becomes more efficient, leading to fewer false negatives. While the LSTM model (Default) produces more false positives overall, it maintains a reasonable ratio of false negatives to true positives. The tuned LSTM model demonstrates an improvement in recall by marginally reducing the amount of false negatives. The modified version of the GRU model significantly reduces false negatives, making it one of the more balanced models,

and the model as a whole shows great performance. As a result of balancing specificity and sensitivity, the Bi-LSTM model (Default) produces the most false positives. Although the Bi-LSTM model has improved accuracy after tuning, it still produces a significant amount of false negatives.

Table 3. Model performance metrics

Model	True positive	False positive	True negative	False negative	Accuracy	Recall	Precision	F1-score
RNN (Default)	892	620	2959	902	0.716732	0.497213	0.589947	0.539625
RNN (Tuned)	1090	630	2949	704	0.751722	0.607581	0.633721	0.620376
LSTM (Default)	1104	696	2883	690	0.742044	0.615385	0.613333	0.614357
LSTM (Tuned)	1116	662	2917	678	0.750605	0.622074	0.627672	0.62486
GRU (Default)	1129	577	3002	665	0.768844	0.62932	0.661782	0.645143
GRU (Tuned)	1181	602	2977	613	0.773869	0.658305	0.662367	0.66033
Bi-LSTM (Default)	1149	718	2861	645	0.746324	0.640468	0.615426	0.627697
Bi-LSTM (Tuned)	1093	541	3038	701	0.768844	0.609253	0.668911	0.63769

Looking at it analytically, the RNN model may miss real security concerns because to its default configuration's increased frequency of false negatives, which makes it less successful in situations where recall is critical. When it is crucial to minimize both false positives and false negatives, the LSTM and GRU models especially in their tuned versions strike a superior balance between recall and precision. Although the Bi-LSTM model's specificity is enhanced after tuning, it may not be able to detect all positive cases because to its relatively high false negative rate. If reducing false negatives is more important than minimizing false positives, as is the case with the Bi-LSTM, then the best model to use would be the one that best suits the application's demands. To make sure all possible threats are identified and handled properly, for example, it may be more important to minimize false negatives in software security.

4.3. ROC – AUC result

Tuning clearly improves each model's AUC-ROC performance, as seen in Figure 3. In the case of sequential security patch data, the GRU model consistently earns the greatest AUC-ROC scores, whether in its default or customized version. This suggests that it is very capable of efficiently distinguishing between classes. The RNN model's performance is greatly improved by tuning, showcasing its improved capacity to detect important patterns in the data and decrease classification mistakes. Tuning also improves AUC-ROC for LSTM and Bi-LSTM models, though to a lesser extent than for GRU and RNN. Especially for complicated tasks with recurrent and sequential data, these findings show that hyperparameter adjustment is crucial for improving model performance. Tuning yields varying results for different models, highlighting the need for a dataset-specific, individualized strategy.

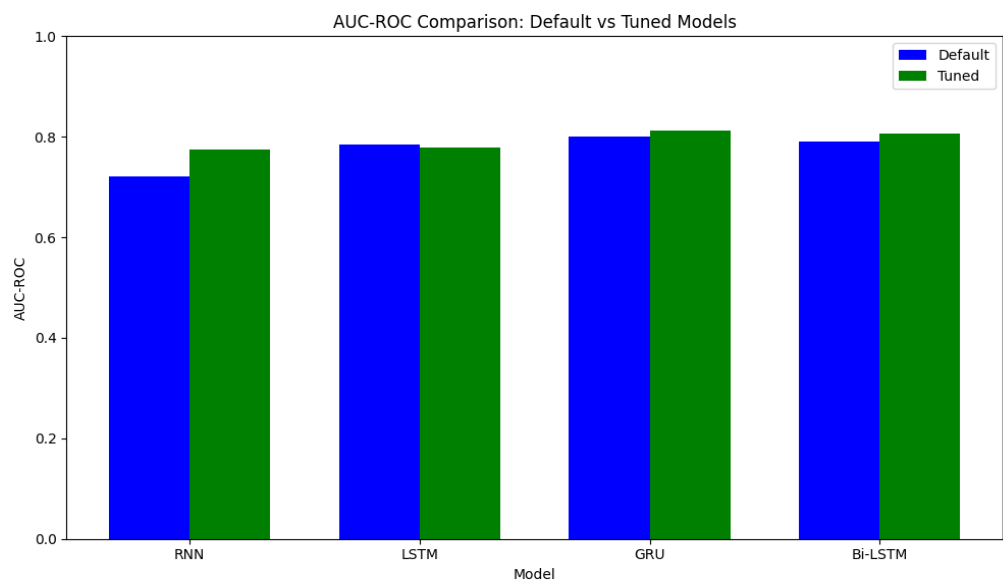


Figure 3. Bar chart of ROC-AUC result

4.4. Discussion

With hyperparameter adjustment in particular, the experimental findings show that the GRU model attained the best accuracy and showed balanced performance across all evaluation measures, including F1-score, precision, recall, and accuracy. This indicates that GRU model is well suited for this classification job since it is very good at capturing the patterns and temporal dependencies that are inherent in sequential and recurrent security patch data. To ensure that major security vulnerabilities are discovered immediately, GRUs accurately detect trends across many patches deployed over time by modelling such temporal linkages. It is critical to not miss any security risks, particularly recurrent ones, and the tuned RNN model shown considerable improvement in this area, particularly in lowering false negatives. On the other side, the LSTM and Bi-LSTM models produced more false negatives despite keeping specificity high. This suggests that they are more cautious and may overlook some security updates, especially those that are part of a consecutive release. This compromise emphasizes the significance of selecting a model according to the unique properties of the security patch data and the requirements of the application. A model like as the tweaked GRU, which provides a balanced way to capture both short-term and long-term data dependencies, could be better in situations where the repercussions of missing a possible danger are high. The results show that tweaking hyperparameters is crucial for improving model performance, especially with recurrent and sequential data, because all models improved significantly. There are a lot of real-world applications for these fine-tuned models when it comes to identifying security patches. One area where they could be useful is in software systems, where the release order and timing of patches are very important for vulnerability management.

5. CONCLUSION

Finally, this work set out to use deep learning techniques to investigate and resolve issues related to sequential and recurrent security patch data. The research successfully found the best suited deep learning models for this complicated task by executing a series of carefully prepared experiments. These trials included detailed hyperparameter tuning and model evaluation. In terms of capturing complex temporal correlations within security patch data and maintaining balanced performance across important metrics including accuracy, precision, recall, and F1-score, the GRU model proved to be the most effective among those that were evaluated. Because it guarantees both high sensitivity and specificity two qualities that are critical for correctly detecting and categorizing security patches the GRU model is especially useful in handling sequential and recurrent data due to its balanced performance. The trade-off between minimizing false positives and avoiding the omission of actual threats was highlighted by models like LSTM and Bi-LSTM, which showed higher specificity but produced more false negatives, even though they could process extensive temporal sequences. Because various scenarios may place a premium on different aspects of performance, our results highlight the need of application-specific model selection and hyperparameter adjustment. This study sheds light on deep learning's potential in cybersecurity by solving the fundamental problem of categorizing complicated, sequential security patch data. Deploying well-optimized deep learning models can greatly improve software system security by making vulnerability identification and patch management processes more efficient and less requiring human interaction. This has far-reaching practical ramifications. Adding more types of data to the dataset, putting these models through their paces in real-world scenarios, and exploring hybrid models that draw from different deep learning architectures should all be goals of future studies. In addition, by utilizing external data sources or incorporating contextual information, advanced feature engineering techniques can enhance these models' performance and adaptability. This means they can be applied to a wider range of cybersecurity challenges.

ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to all individuals who provided technical and editorial support throughout this research.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Falah Muhammad Alam	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Devi Fitriannah	✓	✓		✓		✓				✓		✓		

C : Conceptualization	I : Investigation	Vi : Visualization
M : Methodology	R : Resources	Su : Supervision
So : Software	D : Data Curation	P : Project administration
Va : Validation	O : Writing - Original Draft	Fu : Funding acquisition
Fo : Formal analysis	E : Writing - Review & Editing	

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are openly available in the PatchDB repository at <https://github.com/SunLab-GMU/PatchDB>.

REFERENCES

[1] Y. Zhou, J. K. Siow, C. Wang, S. Liu, and Y. Liu, "SPI: Automated identification of security patches via commits," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 1, pp. 1–27, Jan. 2022, doi: 10.1145/3468854.

[2] T. Ganz, E. Imgrund, M. Härterich, and K. Rieck, "PAVUDI: Patch-based vulnerability discovery using machine learning," in *Annual Computer Security Applications Conference*, Dec. 2023, pp. 704–717, doi: 10.1145/3627106.3627188.

[3] X. Wang *et al.*, "PatchRNN: A deep learning-based system for security patch identification," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, Nov. 2021, pp. 595–600, doi: 10.1109/MILCOM52596.2021.9652940.

[4] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, Oct. 2023, pp. 654–668, doi: 10.1145/3607199.3607242.

[5] B. Steenhoek, H. Gao, and W. Le, "Dataflow analysis-inspired deep learning for efficient vulnerability detection," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, Feb. 2024, pp. 1–13, doi: 10.1145/3597503.3623345.

[6] M. S. Ansari, V. Bartoš, and B. Lee, "GRU-based deep learning approach for network intrusion alert prediction," *Future Generation Computer Systems*, vol. 128, pp. 235–247, Mar. 2022, doi: 10.1016/j.future.2021.09.040.

[7] G. S. Chadha, A. Panambilly, A. Schwung, and S. X. Ding, "Bidirectional deep recurrent neural networks for process fault classification," *ISA Transactions*, vol. 106, pp. 330–342, Nov. 2020, doi: 10.1016/j.isatra.2020.07.011.

[8] S. Y.-C. Chen, S. Yoo, and Y.-L. L. Fang, "Quantum long short-term memory," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 8622–8626, doi: 10.1109/ICASSP43922.2022.9747369.

[9] R. K. Sharma and M. Casas, "Task-based acceleration of bidirectional recurrent neural networks on multi-core architectures," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2022, pp. 941–951, doi: 10.1109/IPDPS53621.2022.00096.

[10] T. Ahmad, J. Wu, H. S. Alwageed, F. Khan, J. Khan, and Y. Lee, "Human activity recognition based on deep-temporal learning using convolution neural networks features and bidirectional gated recurrent unit with features selection," *IEEE Access*, vol. 11, pp. 33148–33159, 2023, doi: 10.1109/ACCESS.2023.3263155.

[11] Z. Wei, Q. Zhu, C. Min, Y. Chen, and G. Wang, "Bidirectional hybrid LSTM based recurrent neural network for multi-view stereo," *IEEE Transactions on Visualization and Computer Graphics*, vol. 30, no. 7, pp. 3062–3073, Jul. 2024, doi: 10.1109/TVCG.2022.3165860.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: a next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul. 2019, pp. 2623–2631, doi: 10.1145/3292500.3330701.

[13] B. Lim, S. Zohren, and S. Roberts, "Population-based global optimisation methods for learning long-term dependencies with RNNs," *arXiv preprint arXiv:1905.09691*, 2019.

[14] K. Joshi, R. P. Pothukuchi, A. Wibisono, and A. Bhattacharjee, "Mitigating catastrophic forgetting in long short-term memory networks," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2023.

[15] Y. Zhang, "Encoder-decoder models in sequence-to-sequence learning: A survey of RNN and LSTM approaches," *Applied and Computational Engineering*, vol. 22, no. 1, pp. 218–226, Oct. 2023, doi: 10.54254/2755-2721/22/20231220.

[16] S. Mohsen, "Recognition of human activity using GRU deep learning algorithm," *Multimedia Tools and Applications*, vol. 82, no. 30, pp. 47733–47749, Dec. 2023, doi: 10.1007/s11042-023-15571-y.

[17] X. Wang, S. Wang, P. Feng, K. Sun, and S. Jajodia, "PatchDB: A large-scale security patch dataset," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2021, pp. 149–160, doi: 10.1109/DSN48987.2021.00030.

[18] A. Caciularu, I. Dagan, and J. Goldberger, "Denoising word embeddings by averaging in a shared space," in *Proceedings of *SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, 2021, pp. 294–301, doi: 10.18653/v1/2021.starsem-1.28.




[19] P. Boumis *et al.*, "Deep optical study of the mixed-morphology supernova remnant G 132.7+1.3 (HB3)," *Monthly Notices of the Royal Astronomical Society*, vol. 512, no. 2, pp. 1658–1676, Mar. 2022, doi: 10.1093/mnras/stac412.

[20] S. Li, F. Bielsa, M. Stock, A. Kiss, and H. Fang, "An investigation of magnetic hysteresis error in Kibble balances," *IEEE*




- Transactions on Instrumentation and Measurement*, vol. 69, no. 8, pp. 5717–5726, Aug. 2020, doi: 10.1109/TIM.2019.2962848.
- [21] A. Makinde, “Optimizing time series forecasting: a comparative study of Adam and nesterov accelerated gradient on LSTM and GRU networks using stock market data,” *arXiv preprint arXiv:2410.01843*, 2024, [Online]. Available: <http://arxiv.org/abs/2410.01843>.
 - [22] M. M., “Ensemble of classifiers in text categorization,” *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 1, pp. 46–53, Jan. 2020, doi: 10.30534/ijeter/2020/09812020.
 - [23] R. Shinde and C. Kalpana, “Advancements in deep learning: A comprehensive review,” *REST Journal on Data Analytics and Artificial Intelligence*, vol. 2, no. 2, Jun. 2023, doi: 10.46632/jdaai/2/2/7.
 - [24] J. M. Johnson and T. M. Khoshgoftaar, “Survey on deep learning with class imbalance,” *Journal of Big Data*, vol. 6, no. 1, p. 27, Dec. 2019, doi: 10.1186/s40537-019-0192-5.
 - [25] H. Holzmann and B. Klar, “Robust performance metrics for imbalanced classification problems,” in *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, 2024, pp. 245–251.
 - [26] A. Darmawahyuni, S. Nurmaini, M. N. Rachmatullah, F. Firdaus, and B. Tutuko, “Unidirectional-bidirectional recurrent networks for cardiac disorders classification,” *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 19, no. 3, pp. 902–910, Jun. 2021, doi: 10.12928/telkomnika.v19i3.18876.
 - [27] F. Movahedi, R. Padman, and J. F. Antaki, “Limitations of receiver operating characteristic curve on imbalanced data: Assist device mortality risk scores,” *The Journal of Thoracic and Cardiovascular Surgery*, vol. 165, no. 4, pp. 1433–1442.e2, Apr. 2023, doi: 10.1016/j.jtcvs.2021.07.041.
 - [28] J. Miao and W. Zhu, “Precision–recall curve (PRC) classification trees,” *Evolutionary Intelligence*, vol. 15, no. 3, pp. 1545–1569, Sep. 2022, doi: 10.1007/s12065-021-00565-2.
 - [29] F. V. Stoye, C. Tschammler, O. Kuss, and A. Hoyer, “A discrete time-to-event model for the meta-analysis of full ROC curves,” *Research Synthesis Methods*, vol. 15, no. 6, pp. 1031–1048, Nov. 2024, doi: 10.1002/jrsm.1753.
 - [30] A. Das *et al.*, “Automatic error analysis for document-level information extraction,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 3960–3975, doi: 10.18653/v1/2022.acl-long.274.
 - [31] K. M. Beyene and A. El Ghouch, “Time-dependent ROC curve estimation for interval-censored data,” *Biometrical Journal*, vol. 64, no. 6, pp. 1056–1074, Aug. 2022, doi: 10.1002/bimj.202000382.
 - [32] B. Gajic, A. Amato, R. Baldrich, J. van de Weijer, and C. Gatta, “Area under the ROC curve maximization for metric learning,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun. 2022, pp. 2806–2815, doi: 10.1109/CVPRW56347.2022.00318.

BIOGRAPHIES OF AUTHORS



Falah Muhammad Alam    received his bachelor's degree (S.Kom) in cyber security from Binus University in 2020, where he developed a strong foundation in protecting information systems against cyber threats. Currently, he is pursuing a master's degree (M.TI) in cyber security at the same institution, focusing on innovative solutions to emerging challenges in the field. With three years of experience in the IT Department of a leading banking company, Falah oversees the security of critical information systems, ensuring compliance with industry regulations, and implementing robust cybersecurity measures. His professional role has provided him with hands-on experience in tackling real-world security challenges, particularly in the financial sector. His research interests lie at the intersection of cyber security and data science, where he explores the use of advanced analytics to enhance threat detection. He can be reached at falah.alam@binus.ac.id.



Devi Fitrianah    is a lecturer and researcher at the Master of Computer Science at Bina Nusantara University. She received her bachelor's degree in computer science from Bina Nusantara University in 2000 and a master's degree in information technology and a Ph.D. degree in computer science from Universitas Indonesia in 2008 and 2015 respectively. She has been rewarded with a sandwich program at the Laboratory for Pattern Recognition and Image Processing and GIS (PRIPGIS Lab) Department of Computer Science, Michigan State University, East Lansing, Michigan, USA in 2014. She can be contacted via email: devi.fitrianah@binus.ac.id.