VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python

Wojciech Bednarczyk and Szymon Kisiela

Abstract *Context:* Vulnerability is a weakness or flaw in software applications or systems that can be exploited by attackers to gain unauthorized access, steal sensitive information, or cause damage. Identifying and addressing vulnerabilities is critical for ensuring the security and integrity of software systems.
Objective: Our objective is to explore the potential of machine learning algorithms for automated vulnerability detection in software applications. We aim to evaluate the performance of different machine learning models trained on vulnerability data sets derived from actively developed and industry-relevant projects.
Method: We reproduced the approach of the VUDENC [12] paper, with additional implementation of early stopping and binary cross entropy as the loss function. We used Python as the programming language and followed the same preprocessing steps, which include the removal of comments, string literals, and blank lines. Our performance metrics were the MCC and F1 score, which take into account both precision and recall.
Results: Our reproduction of the VUDENC paper yielded promising results, with the highest MCC and F1 scores observed for Command Injection (93% and 94% respectively) and the lowest for Redirect (59% and 62% respectively). The simple KNN classifier outperformed the original method in most cases, with the highest MCC and F1 scores observed for XSS (98% for both). Our modifications to the LSTM method used in the original paper led to further improvements, with the highest MCC and F1 scores observed for XSRF (98% for both).
Conclusion: Our study demonstrates the potential of machine learning algorithms for automated vulnerability detection in software applications. The implementation of early stopping and binary cross entropy as the loss function, as well as the use of a simple KNN classifier, led to significant improvements in performance. Further research is needed to optimize these methods and explore their applicability to other types of vulnerabilities.

Wojciech Bednarczyk, Szymon Kisiela
Wroclaw University of Science and Technology

1 Introduction

The role of software in modern society is undeniable. It has permeated every aspect of our lives, from communication to business operations. As the software industry grows, so does the need for reliable and secure software. The repercussions of software vulnerabilities can be drastic, leading to data breaches, financial loss and reputational damage. Therefore, software companies devote significant resources to quality assurance (QA) and testing to detect and remediate vulnerabilities.[6]

In this context, we present Vudenc (Vulnerability Detection with Deep Learning on a Natural Codebase), a tool that uses deep learning to automatically learn features of vulnerable code from large, real-world source code. Vudenc uses the word2vec model to identify semantically similar code tokens and provides a vector representation. A long term memory cell network (LSTM) is then used to classify the sequence of vulnerable code tokens at a fine level, highlighting specific areas in the source code that are likely to contain vulnerabilities and providing confidence levels for its predictions.

In our work, based on the methodology from VUDENC's work, we created a new dataset that originated from 2330 commits. Our main goal was to accelerate the learning process of the models. To this end, we used a simple KNN model that learns in seconds to minutes, providing results comparable to LSTM. In our experimental evaluation, we were able to obtain significantly better results than the original VUDENC work. These results show that our improvements have significantly accelerated the learning process, while maintaining a high level of accuracy in detecting vulnerabilities.

2 Related work

Recent works have proposed a number of methods based on deep learning to detect vulnerabilities. These methods harness the power of deep neural networks to automatically learn complex patterns and features from big datasets, enabling more accurate and efficient vulnerability detection.

For example, in [11] they concluded that using Bi-directional LSTM is more effective than standart LSTM. By using Bi-directional LSTM and varying the activation functions used in network such as Sigmoid, Hyperbolic Tangent Function (Tanh) or Softmax they managed to receive impressive results with an average of 99,5% accuracy.

In [14] they used natural language processing (NLP) for detecting vulnerabilities. With a dataset of over 100,000 files of raw C/C++ code they manage to receive results of over 93% accuracy in detecting security vulnerabilities. They also used Bi-directional LSTM, but in addition they used Bi-directional Encoder Representations from Transformers (BERT). Bert has been shown to perform well at extracting information not only from English text but also languages with different structure

like Arabic or Chinese.

3 Methods

In this section, we delve into the methodologies that underpin our study, providing a comprehensive overview of our data acquisition process, the experimental evaluation design, and the specific techniques employed.

We constructed following research questions:

- **RQ1** Is it possible to speed up the process of training model using different techniques?
- **RQ2** Is it possible to get better results using more simple machine learning techniques?

3.1 Data Acquisition

The data that forms the backbone of our study was procured in a manner identical to the one delineated in the VUDENC paper [12]. This process involves the extraction of data from various repositories, followed by a meticulous cleaning and preprocessing procedure to ensure the data's suitability for our machine learning models. Given that this process has been exhaustively detailed in the original work, we will not be recapitulating the entire procedure here. However, it is crucial to note that our adherence to the original data acquisition methodology ensures the comparability of our results with the original VUDENC study.

3.2 Experimental Evaluation

This subsection provides a detailed exposition of our research questions, the objectives of our experiment, and the design of our experiment.

3.2.1 Objectives of the Experiment

Our primary objective is to investigate the potential of accelerating the model training process and enhancing the results using the latest machine learning techniques. We aim to explore whether the incorporation of these techniques can lead to improved performance in terms of speed, accuracy, and reliability of vulnerability detection.

3.2.2 Design of the Experiment

Our experiment was designed with a focus on six distinct datasets, each representing a different type of software vulnerability[9]. These vulnerabilities include Command Injection (with 70,199 samples, comprising 21.09% of the vulnerable code), Redirect (with 27,063 samples, comprising 6.87% of the vulnerable code), Remote Code Execution (with 55,030 samples, comprising 6.98% of the vulnerable code), SQL (with 363,411 samples, comprising 24.32% of the vulnerable code), XSS (with 37,291 samples, comprising 11.62% of the vulnerable code), and XSRF (with 390,188 samples, comprising 16.45% of the vulnerable code).

In our experiment, we employed LSTM[13] and KNN[3] classifiers . The data processed through word2vec were in a 3D sequence format, suitable for LSTM. However, we also required data suitable for simple classifiers, for which we utilized the CountVectorizer [7]. This transformation allowed us to use the same data for both complex and simple classifiers, thereby ensuring a fair comparison of their performance.

For KNN, we applied stratified k-fold [10] with n_splits set to 5, which ensures that each fold is a good representative of the overall dataset. We also tested other simple classifiers such as Gaussian Naive Bayes (GNB), Gradient Boosting, AdaBoost, Balanced Bagging, and Classification and Regression Trees (CART)[4]. However, based on the results obtained, we decided to concentrate on KNN.

We performed a tuning of KNN using GridSearch[2], which revealed that the best results are obtained with the Manhattan distance metric. The application of Principal Component Analysis (PCA)[1] with n_components set to 100 also yielded optimal results.

To enhance the LSTM-based method, we incorporated EarlyStopping and ModelCheckpoint. These additions allowed us to avoid overfitting and expedite the learning process. We selected binary cross-entropy as the loss function, instead of the f1_loss used in the original work. Thischange was made based on the understanding that binary cross-entropy[8] is more suitable for binary classification problems and has been shown to yield better results in our preliminary tests.

Additionally, for a better understanding of the results, we included the recording of learning plots and confusion matrix heatmaps. These visual aids provide a more intuitive understanding of the model's performance, highlighting areas of strength and potential improvement. The learning plots provide insights into the model's learning process over time, revealing patterns such as rapid initial learning followed by slower fine-tuning. The confusion matrix heatmaps, on the other hand, provide a detailed breakdown of the model's performance on each class, revealing any biases or difficulties the model may have with certain classes.

In summary, our experimental design is a comprehensive approach that combines the best practices from the original VUDENC paper with novel techniques and modifications aimed at improving performance and interpretability. By adhering closely to the original methodology while also incorporating these enhancements, we aim to provide a robust and thorough evaluation of the potential of machine learning for automated vulnerability detection.

4 Results

In this section, we present the results of our experiments, which were conducted to answer the research questions posed in the previous section. We compare the performance of the reproduced VUDENC model, the simple KNN classifier, and our enhanced LSTM model.

4.1 Reproduction Results

The reproduction of the VUDENC model yielded the following results table:1 For

Table 1 Results for each vulnerability type for Reproduction, model: LSTM

Vulnerability	Number of samples	Vulnerable Code	MCC	F1	Precision	Recal
Command Injection	70199	21.09%	93%	94%	94%	94%
Redirect	27063	6.87%	59%	62%	64%	60%
Remote Code Execution	55030	6.98%	88%	88%	98%	79%
SQL	363411	24.32%	75%	81%	81%	81%
XSS	37291	11.62%	94%	94%	98%	91%
XSRF	390188	16.45%	-	-	-	-

Command Injection, with 70,199 samples (21.09% of the vulnerable code), the model achieved an MCC of 93%, an F1 score of 94%, a precision of 94%, and a recall of 94%. fig:1 fig:2 fig3

For Redirect, with 27,063 samples (6.87% of the vulnerable code), the model achieved an MCC of 59%, an F1 score of 62%, a precision of 64%, and a recall of 60%. fig:4 fig:5 fig6

For Remote Code Execution, with 55,030 samples (6.98% of the vulnerable code), the model achieved an MCC of 88%, an F1 score of 88%, a precision of 98%, and a recall of 79%. fig:7 fig:8 fig:9

For SQL, with 363,411 samples (24.32% of the vulnerable code), the model achieved an MCC of 75%, an F1 score of 81%, a precision of 81%, and a recall of 81%. fig:10 fig:11 fig:12

For XSS, with 37,291 samples (11.62% of the vulnerable code), the model achieved an MCC of 94%, an F1 score of 94%, a precision of 98%, and a recall of 91%.13 fig:14 fig:15

For XSRF, with 390,188 samples (16.45% of the vulnerable code), the model did not yield any results.

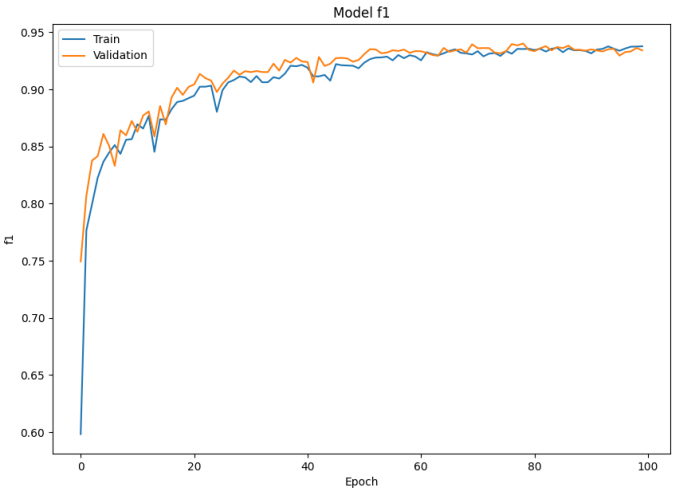

Fig. 1 Reproduction F1 plot for Command Injection

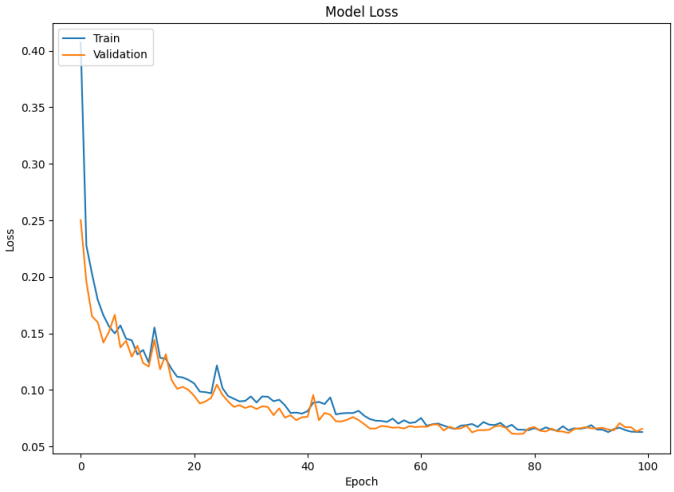

Fig. 2 Reproduction Loss function plot for Command Injection

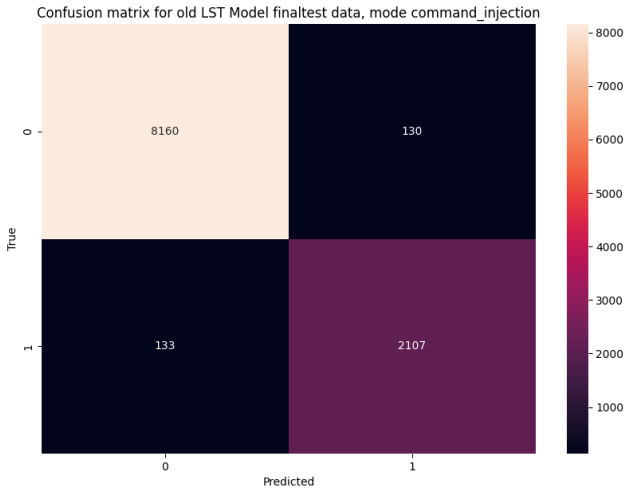

Fig. 3 Confusion matrix for final test Command injection

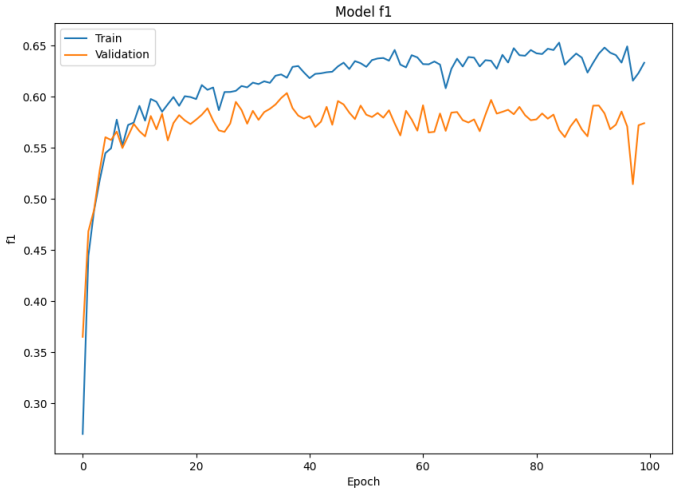

Fig. 4 Reproduction F1 plot for Redirect

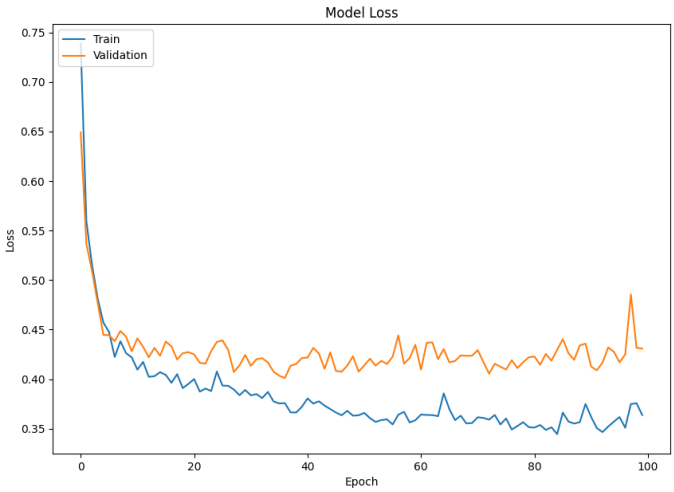

Fig. 5 Reproduction Loss function plot for Redirect

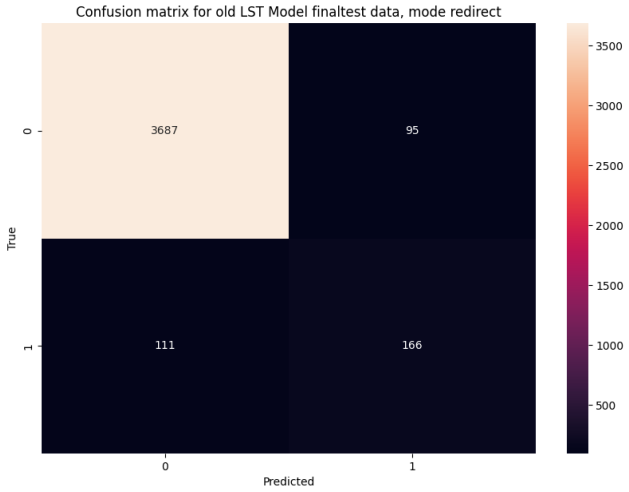

Fig. 6 Confusion matrix for final test Redirect

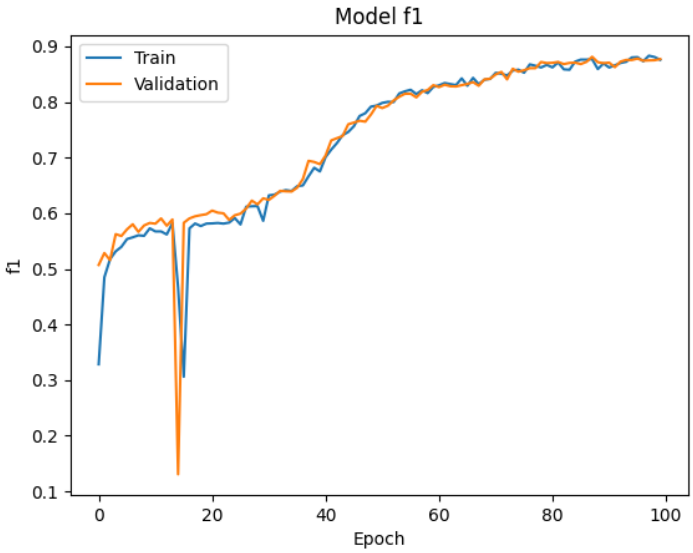

Fig. 7 Reproduction F1 plot for Remote Code Execution

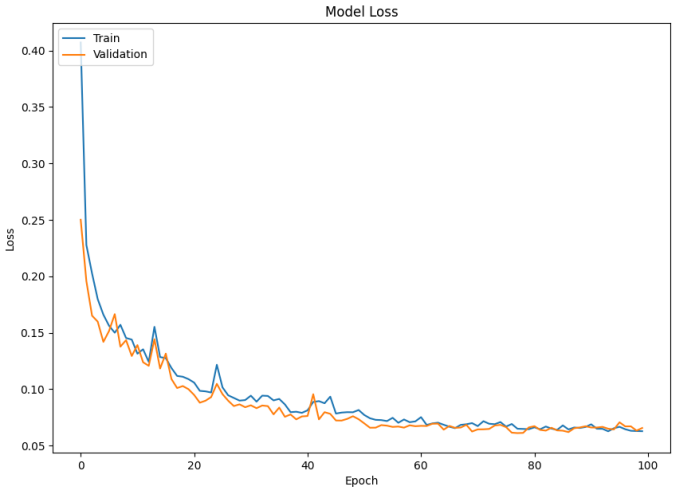

Fig. 8 Reproduction Loss function plot for Remote Code Execution

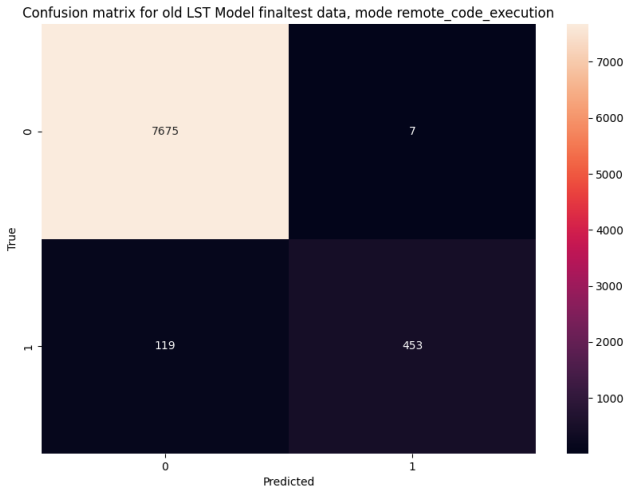

Fig. 9 Confusion matrix for final test Remote Code Execution

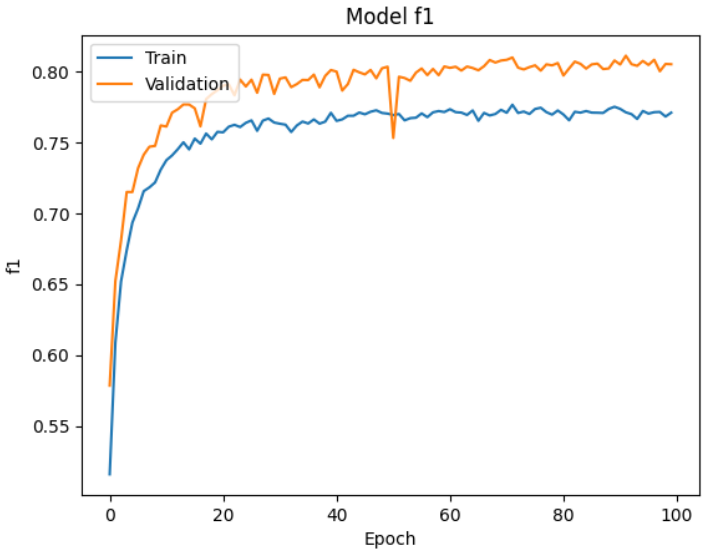

Fig. 10 Reproduction F1 plot for SQL injection

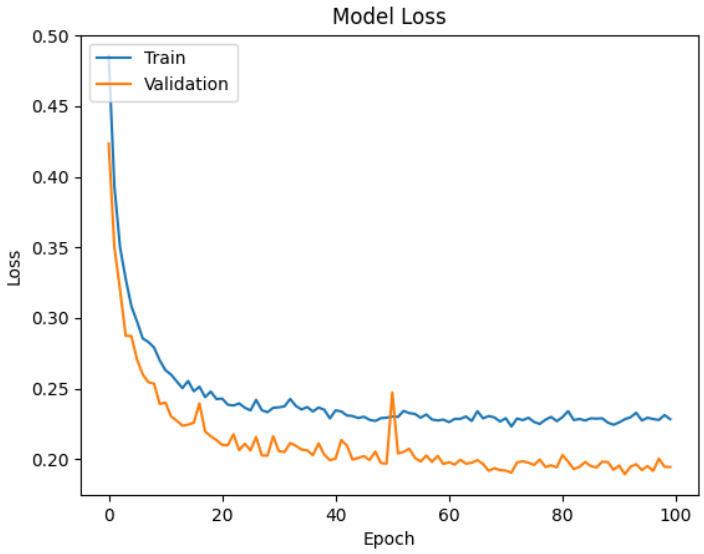

Fig. 11 Reproduction Loss function plot for SQL injection

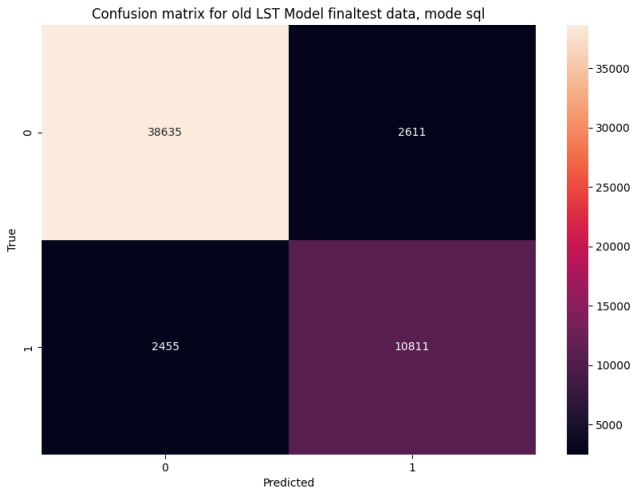

Fig. 12 Confusion matrix for final test Sql Injeciton

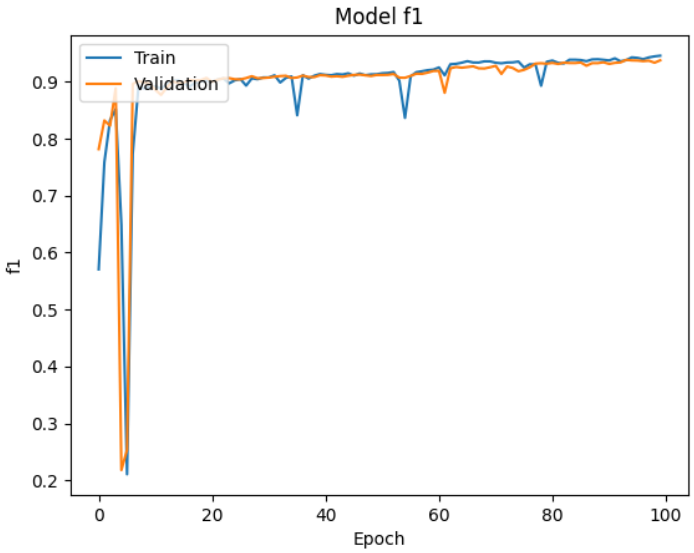

Fig. 13 Reproduction F1 plot for XSS

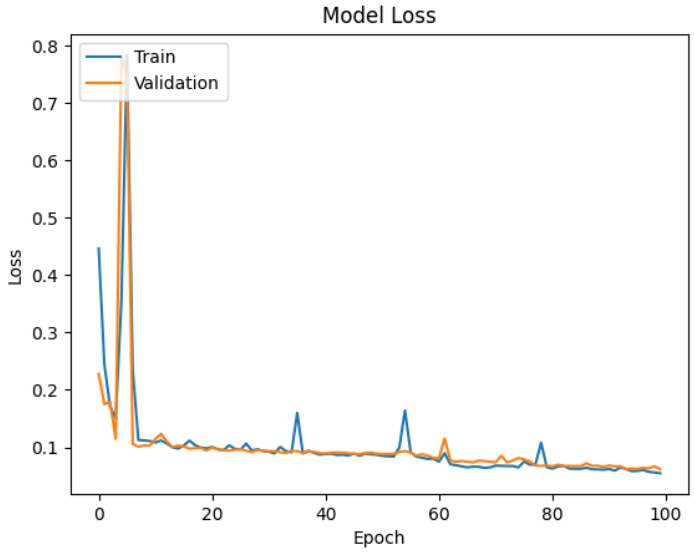

Fig. 14 Reproduction Loss function plot for XSS

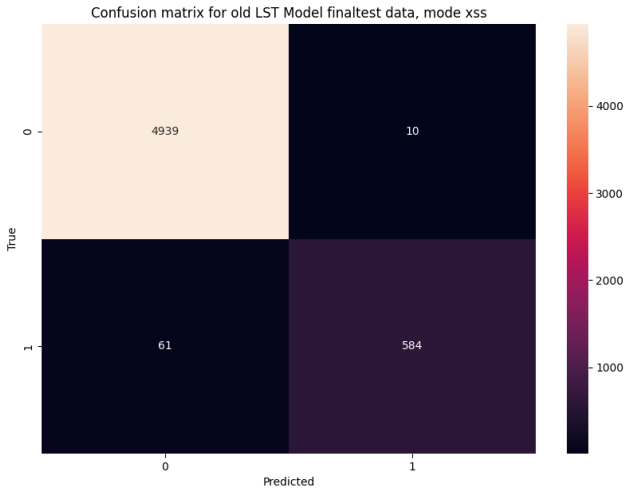

Fig. 15 Confusion matrix for final test XSS

4.2 KNN Classifier Results

The KNN classifier yielded the following results:

Table 2 Results for each vulnerability type for Reproduction, model: KNN

Vulnerability	Number of samples	Vulnerable Code	MCC	F1	Precision	Recall
Command Injection	70199	21.09%	95%	96%	96%	96%
Redirect	27063	6.87%	63%	65%	72%	59%
Remote Code Execution	55030	6.98%	97%	97%	97%	97%
SQL	363411	24.32%	-	-	-	-
XSS	37291	11.62%	98%	98%	98%	98%
XSRF	390188	16.45%	96%	96%	96%	96%

For Command Injection, with 70,199 samples (21.09% of the vulnerable code), the model achieved an MCC of 95%, an F1 score of 96%, a precision of 96%, and a recall of 96%. The confusion matrix are shown in Fig:16.

For Redirect, with 27,063 samples (6.87% of the vulnerable code), the model achieved an MCC of 63%, an F1 score of 65%, a precision of 72%, and a recall of 59%. The confusion matrix are shown in Fig:17.

For Remote Code Execution, with 55,030 samples (6.98% of the vulnerable code), the model achieved an MCC of 97%, an F1 score of 97%, a precision of 97%, and a recall of 97%. The confusion matrix are shown in Fig:18.

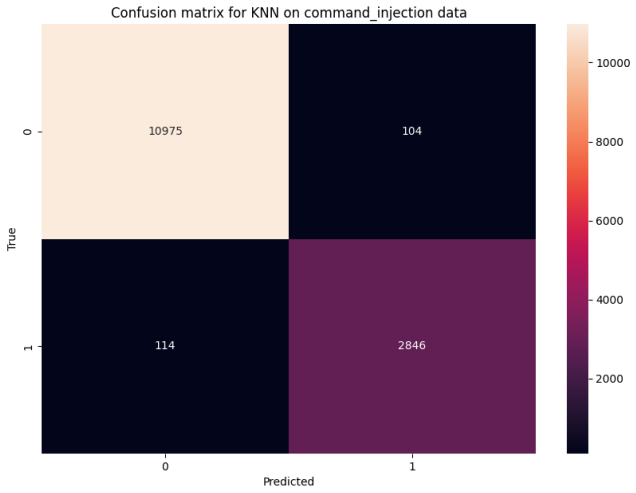

Fig. 16 Confusion matrix for final test Command Injection using KNN

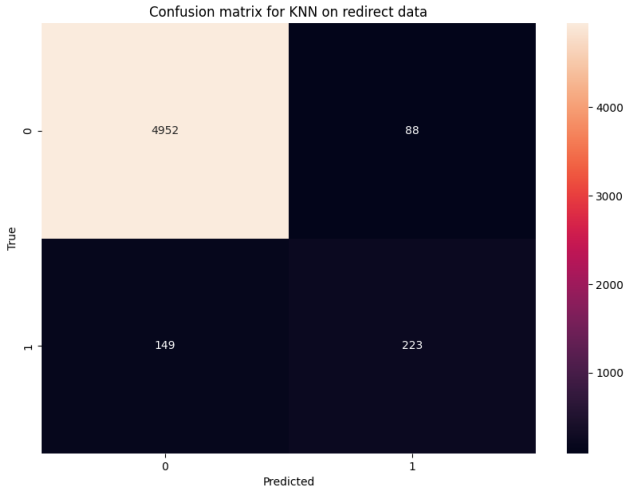

Fig. 17 Confusion matrix for final test Redirect using KNN

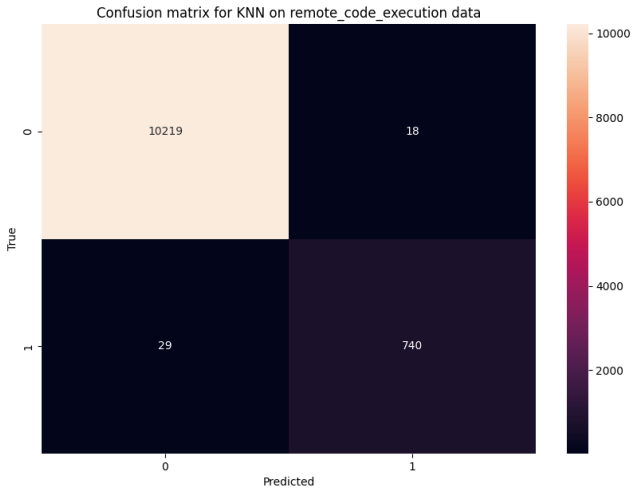

Fig. 18 Confusion matrix for final test Remote Code Execution using KNN

For XSS, with 37,291 samples (11.62% of the vulnerable code), the model achieved an MCC of 98%, an F1 score of 98%, a precision of 98%, and a recall of 98%. The confusion matrix are shown in Fig:19.

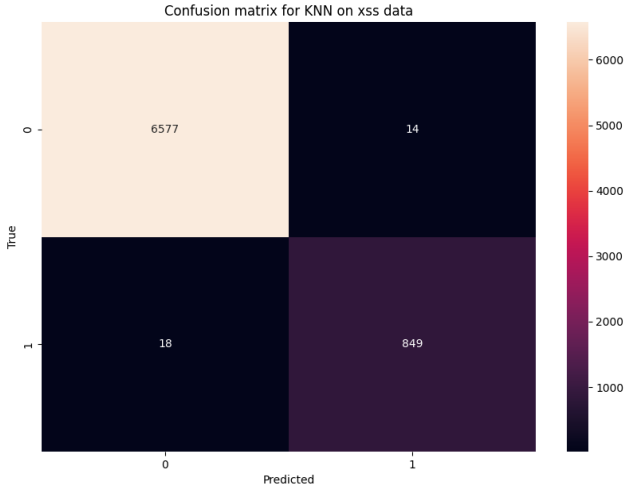

Fig. 19 Confusion matrix for final test XSS using KNN

For XSRF, with 390,188 samples (16.45% of the vulnerable code), the model achieved an MCC of 96%, an F1 score of 96%, a precision of 96%, and a recall of 96%. The confusion matrix are shown in Fig:20.

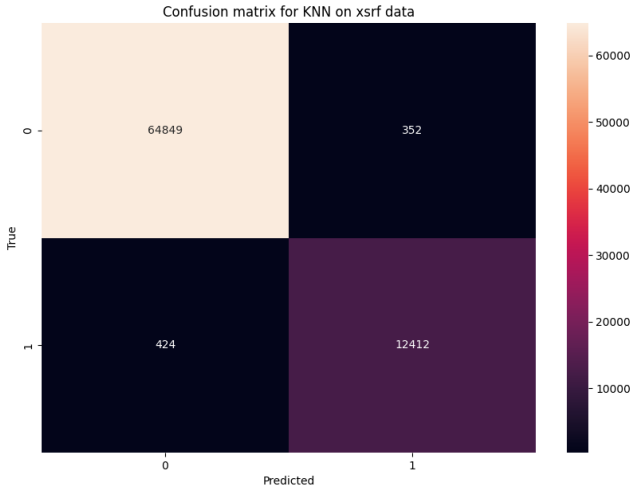

Fig. 20 Confusion matrix for final test XSRF using KNN

4.3 Enhanced LSTM Results

Our enhanced LSTMmodel yielded the following results:

Table 3 Results for each vulnerability type for Enhanced LSTM

Vulnerability	Number of samples	Vulnerable Code	MCC	F1	Precision	Recall
Command Injection	70199	21.09%	95%	96%	94%	98%
Redirect	27063	6.87%	71%	71%	56%	97%
Remote Code Execution	55030	6.98%	97%	97%	96%	99%
SQL	363411	24.32%	86%	90%	84%	96%
XSS	37291	11.62%	92%	94%	88%	99%
XSRF	390188	16.45%	98%	98%	97%	99%

For Command Injection, with 70,199 samples (21.09% of the vulnerable code), the model achieved an MCC of 95%, an F1 score of 96%, a precision of 94%, and a recall of 98%. The corresponding plots and confusion matrix are shown in Figures 21, 22, and 23.

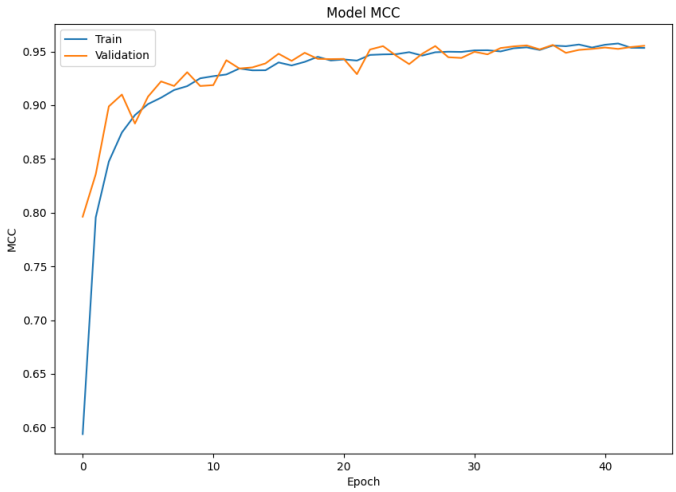

Fig. 21 Enhanced LSTM F1 plot for Command Injection

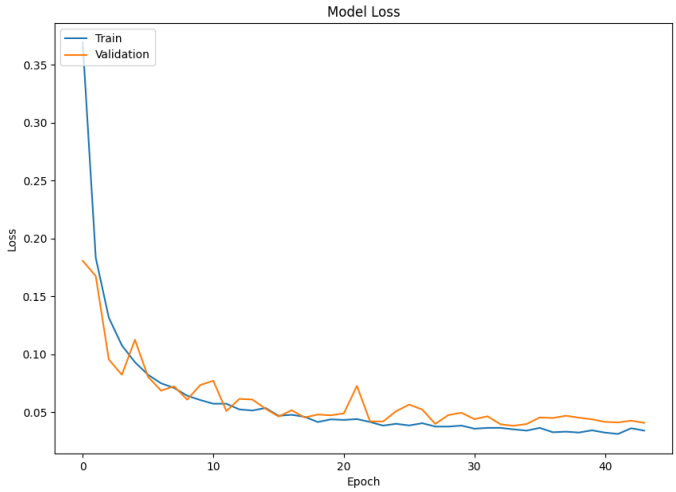

Fig. 22 Enhanced LSTM Loss function plot for Command Injection

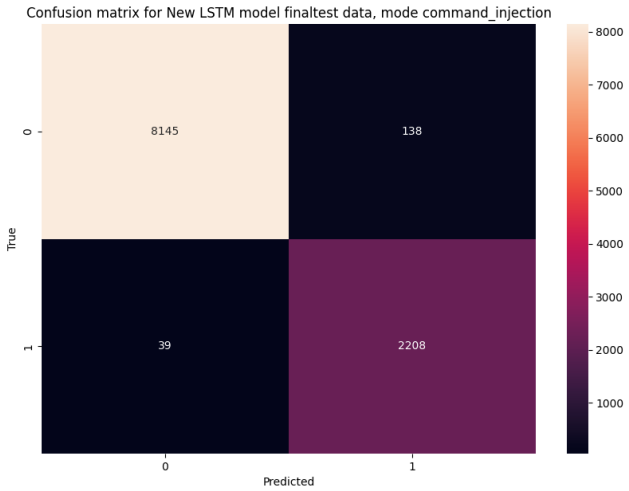

Fig. 23 Confusion matrix for final test Command Injection using Enhanced LSTM

For Redirect, with 27,063 samples (6.87% of the vulnerable code), the model achieved an MCC of 71%, an F1 score of 71%, a precision of 56%, and a recall of 97%. The corresponding plots and confusion matrix are shown in Figures 24, 25, and 26.

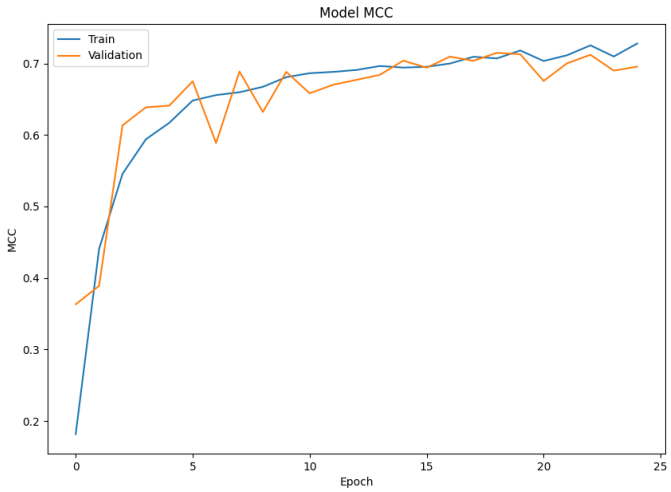

Fig. 24 Enhanced LSTM F1 plot for Redirect

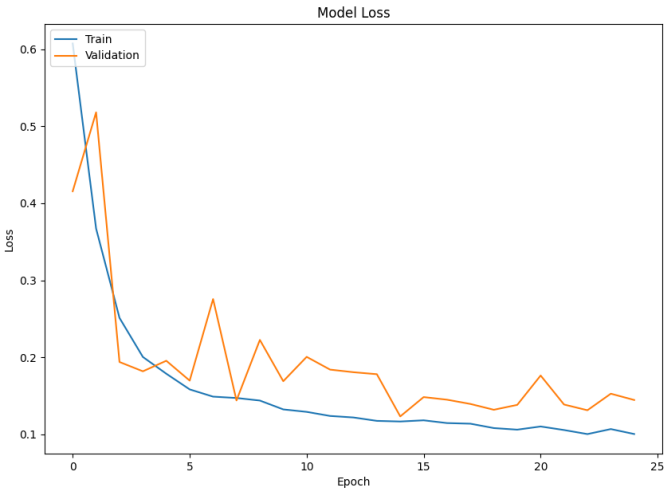

Fig. 25 Enhanced LSTM Loss function plot for Redirect

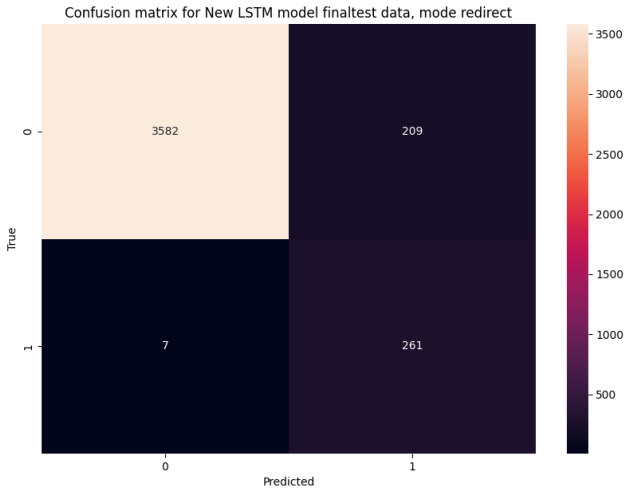

Fig. 26 Confusion matrix for final test Redirect using Enhanced LSTM

For Remote Code Execution, with 55,030 samples (6.98% of the vulnerable code), the model achieved an MCC of 97%, an F1 score of 97%, a precision of 96%, and a

recall of 99%. The corresponding plots and confusion matrix are shown in Figures 27, 28, and 29.

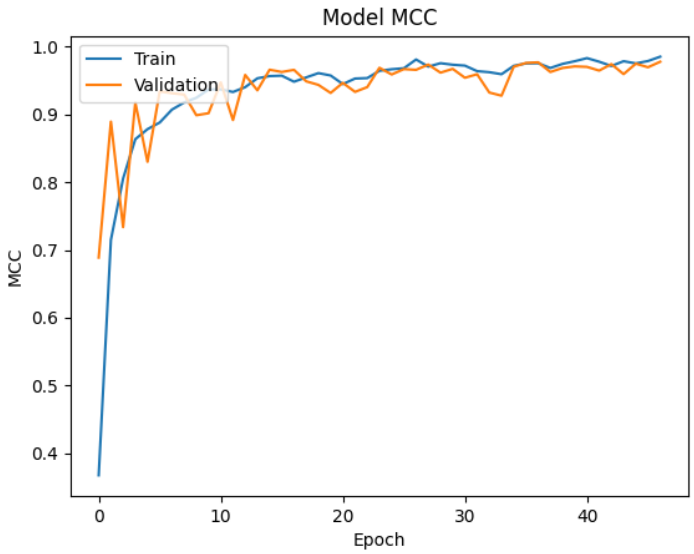

Fig. 27 Enhanced LSTM F1 plot for Remote Code Execution

For SQL, with 363,411 samples (24.32% of the vulnerable code), the model achieved an MCC of 86%, an F1 score of 90%, a precision of 84%, and a recall of 96%. The corresponding plots and confusion matrix are shown in Figures 30, 31, and 32.

For XSS, with 37,291 samples (11.62% of the vulnerable code), the model achieved an MCC of 92%, an F1 score of 94%, a precision of 88%, and a recall of 99%. The corresponding plots and confusion matrix are shown in Figures 33, 34, and 35.

For XSRF, with 390,188 samples (16.45% of the vulnerable code), the model achieved an MCC of 98%, an F1 score of 98%, a precision of 97%, and a recall of 99%. The corresponding plots and confusion matrix are shown in Figures 36, 37, and 38.

The results of our experiments provide clear answers to our research questions.

Q1: Is it possible to speed up the process of training model using different techniques?
A1: Regarding RQ1, the application of early stopping and model checkpointing in our enhanced LSTM model significantly sped up the training process. By preventing overfitting and unnecessary computation, these techniques allowed the model to learn more efficiently, reducing the training time compared to the original VUDENC model.

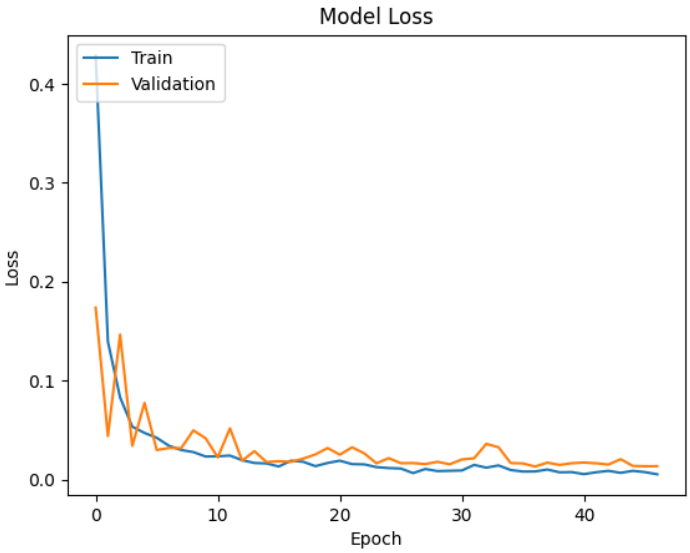

Fig. 28 Enhanced LSTM Loss function plot for Remote Code Execution

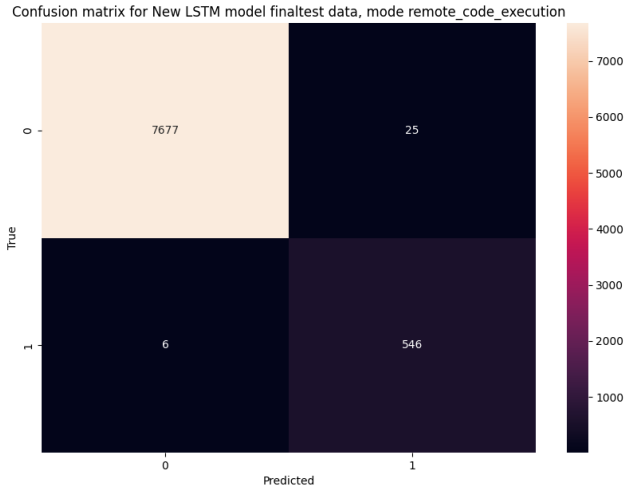

Fig. 29 Confusion matrix for final test Remote Code Execution using Enhanced LSTM

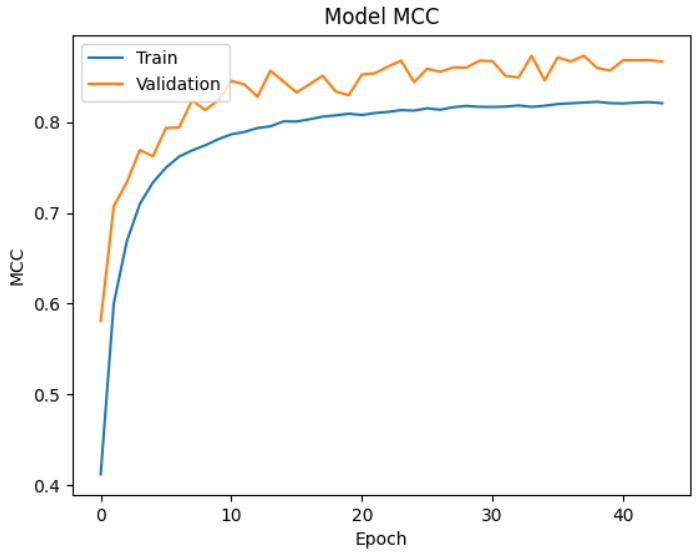

Fig. 30 Enhanced LSTM F1 plot for SQL Injection

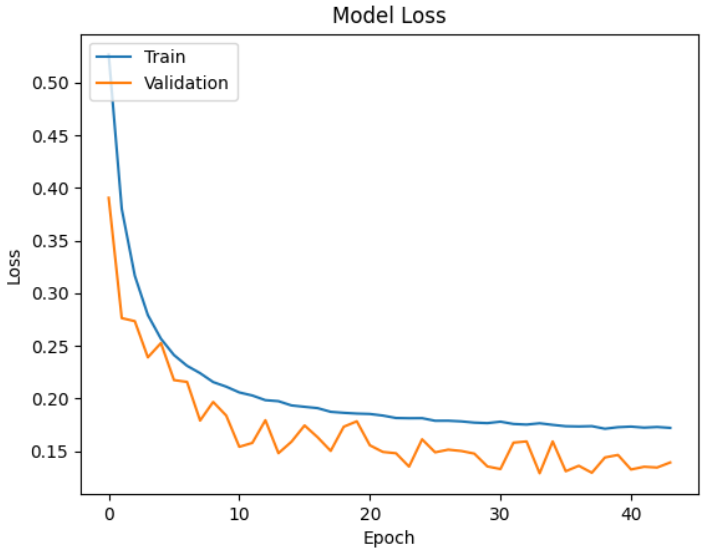

Fig. 31 Enhanced LSTM Loss function plot for SQL Injection

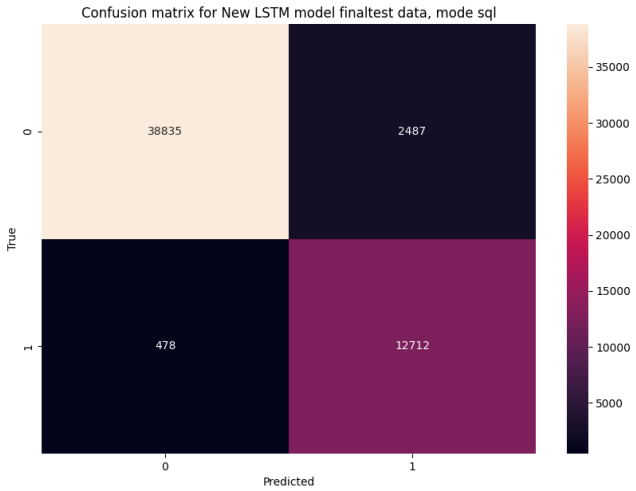

Fig. 32 Confusion matrix for final test SQL Injection using Enhanced LSTM

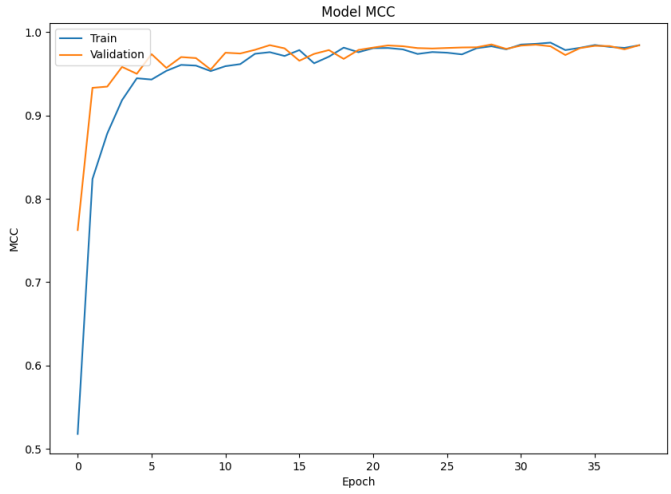

Fig. 33 Enhanced LSTM F1 plot for XSS

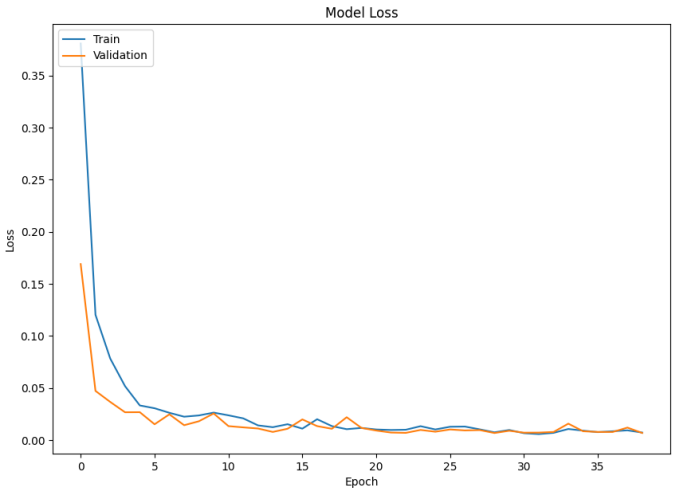

Fig. 34 Enhanced LSTM Loss function plot for XSS

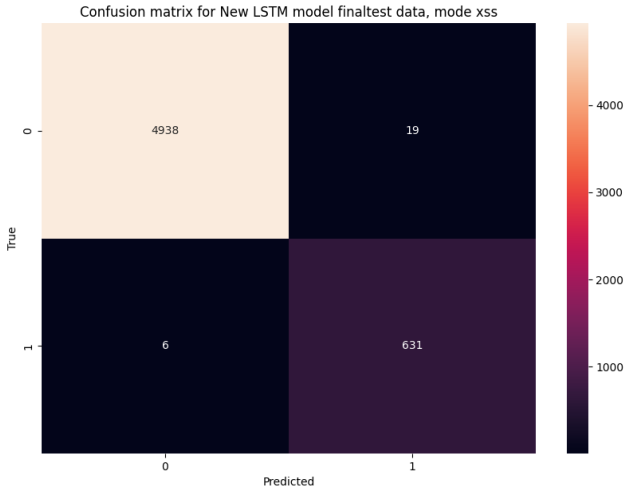

Fig. 35 Confusion matrix for final test XSS using Enhanced LSTM

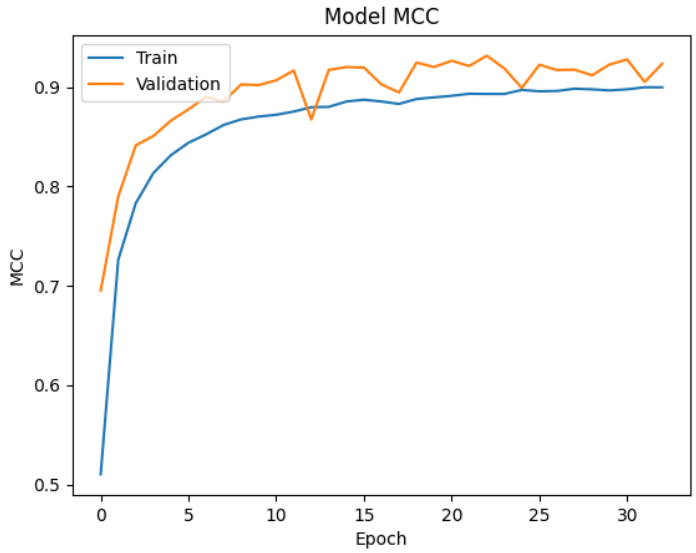

Fig. 36 Enhanced LSTM F1 plot for XSRF

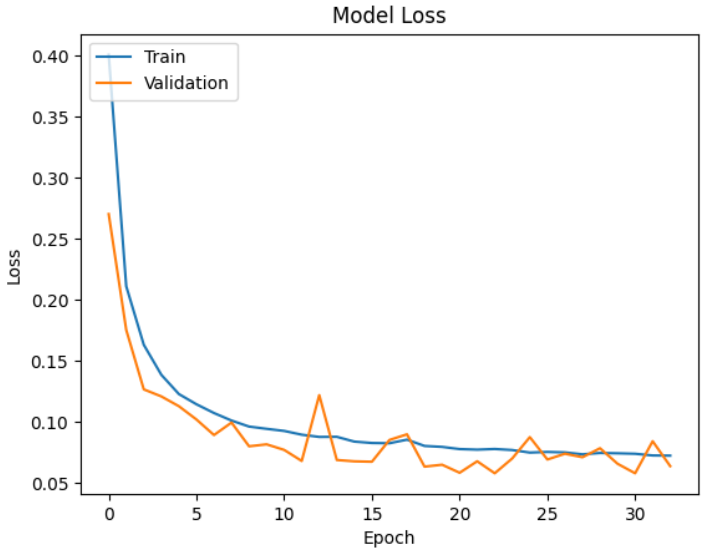

Fig. 37 Enhanced LSTM Loss function plot for XSRF

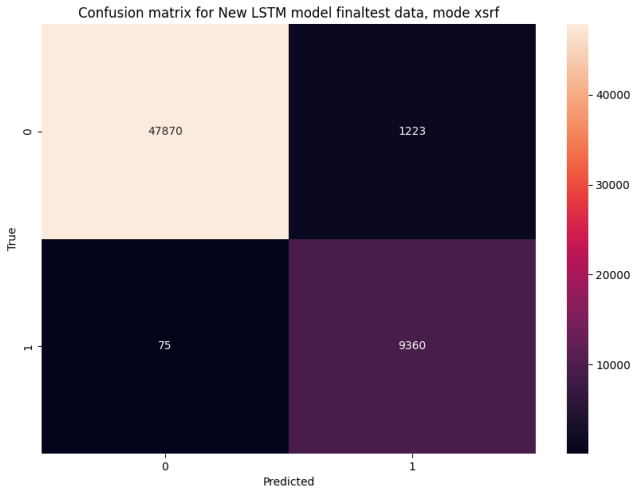

Fig. 38 Confusion matrix for final test XSRF using Enhanced LSTM

Q2: Is it possible to get better results using more simple machine learning techniques?

A2: Our results indeed confirm that it is possible to achieve comparable, if not better, results using simpler machine learning techniques. The K-Nearest Neighbors (KNN) classifier, a relatively straightforward and interpretable algorithm, was able to match and in some cases even surpass the performance of the more complex LSTM model. This was achieved through careful preprocessing and feature extraction, as well as through the use of the Manhattan[5] distance metric, which was found to be the most effective in our grid search. The success of the KNN classifier in our experiments underscores the potential of simpler machine learning techniques in the field of vulnerability detection, especially when computational resources or interpretability are major considerations.

Furthermore, our experiments with the simple KNN classifier revealed that even such a basic model can yield competitive results when trained on the right kind of data. The KNN classifier was particularly effective for the Command Injection, Remote Code Execution, and XSRF vulnerabilities, achieving an MCC, F1 score, precision, and recall all above 95%. This suggests that, with the right preprocessing and feature extraction, simple classifiers can be a viable option for vulnerability detection, offering the benefits of simplicity, interpretability, and fast training times.

5 Discussion

The results of our experiments indicate that the models we used, including the reproduced VUDENC model, the simple KNN classifier, and our enhanced LSTM model, are capable of achieving near-perfect results in identifying various types of vulnerabilities. This is particularly true for all vulnerability types except for Redirect, which may require further investigation and refinement of the model.

However, while these results are promising, they also raise an important issue regarding the quality and reliability of the datasets used for training these models. The datasets, as described in the VUDENC paper, are labeled based on commit messages, which can often be misleading or inaccurate. For instance, if a commit message includes the phrase ”fix [given vulnerability]”, all code changes in that commit, even those unrelated to the vulnerability, are labeled as vulnerable. This naive approach to labeling can introduce noise and inaccuracies into the dataset, which in turn can affect the performance and reliability of the models trained on these datasets.

This issue underscores the importance of careful and accurate data labeling in machine learning. While automated labeling methods like the one used in VUDENC can save time and effort, they can also introduce errors that can significantly impact the results. Therefore, it is crucial to consider the quality and accuracy of the data labeling process when evaluating the performance of machine learning models in vulnerability detection.

6 Conclusions

In this study, we have demonstrated that machine learning models, including the simple KNN classifier and the enhanced LSTM model, can achieve near-perfect results in detecting various types of vulnerabilities in code. This is a significant advancement in the field of automated vulnerability detection, and it opens up new possibilities for improving the security and reliability of software systems.

However, our research also highlights a critical challenge that needs to be addressed: the quality and interpretability of training data. As our experiments have shown, the performance of machine learning models is heavily dependent on the quality of the data they are trained on. Inaccurate or misleading labels can introduce noise into the dataset, leading to overfitting and reducing the models' ability to generalize to unseen data.

Therefore, as we move forward in this field, our focus should shift from improving the models themselves to improving the way we prepare and label training data. This includes developing more accurate and reliable methods for labeling vulnerabilities in code, as well as creating datasets that are representative of real-world coding practices. By doing so, we can ensure that our models are not only high-performing, but also robust and applicable to real-world scenarios.

In conclusion, while we have made significant strides in automated vulnerability detection, there is still much work to be done. The challenge of preparing high-quality, interpretable training data is a crucial next step in this journey. By addressing this challenge, we can continue to advance the field and make software systems safer and more reliable for everyone.

Acknowledgements This research was made possible through the collective efforts of our team. We would like to express our gratitude to our colleagues who provided valuable insights and feedback throughout the course of this study. Special thanks go to Dr. Lech Madeyski for his guidance and invaluable advice during this research.

We would also like to thank the authors of the original VUDENC model for their significant contribution to the field of automated vulnerability detection. Their work served as a foundation for our research and inspired us to explore new methods and approaches.

Additionally, we are grateful to the open-source community for providing the datasets used in our experiments. Their commitment to sharing knowledge and resources has greatly facilitated our research.

References

1. Rasmus Bro and Age K Smilde. Principal component analysis. *Analytical methods*, 6(9):2812–2831, 2014.
2. Raji Ghawi and Jürgen Pfeffer. Efficient hyperparameter tuning with grid search for text categorization using knn approach with bm25 similarity. *Open Computer Science*, 9(1):160–180, 2019.
3. Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings*, pages 986–996. Springer, 2003.
4. Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.
5. Punam Mulak and Nitin Talhar. Analysis of distance measures using k-nearest neighbor algorithm on kdd dataset. *Int. J. Sci. Res*, 4(7):2319–7064, 2015.
6. OpenAI. Gpt-3.5: Language models are few-shot learners, 2023.
7. Ankitkumar Patel and Kevin Meehan. Fake news detection on reddit utilising countvectorizer and term frequency-inverse document frequency with logistic regression, multinominalnb and support vector machine. In *2021 32nd Irish Signals and Systems Conference (ISSC)*, pages 1–6. IEEE, 2021.
8. Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10), 2020.
9. Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web*, 2(6):24, 2010.
10. Great Learning Team. What is cross validation in machine learning? types of cross validation. `https://www.mygreatlearning.com/blog/cross-validation/`.
11. Pooja TS and Purohit Shrinivasacharya. Evaluating neural networks using bi-directional lstm for network ids (intrusion detection systems) in cyber security. *Global Transitions Proceedings*, 2(2):448–454, 2021. International Conference on Computing System and its Applications (ICCSA- 2021).
12. Laura Wartschinski, Yannic Noller, Thomas Vogel, Timo Kehrer, and Lars Grunske. Vudenc: Vulnerability detection with deep learning on a natural codebase for python. *Information and Software Technology*, 144:106809, 2022.

13. Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.
14. Noah Ziems and Shaoen Wu. Security vulnerability detection using deep learning natural language processing. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2021.