Avrupa Bilim ve Teknoloji Dergisi
Sayı 23, S. 686-696, Nisan 2021

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
No. 23, pp. 686-696, April 2021

Research Article

The Role of Vulnerable Software Metrics on Software Maintainability Prediction

Canan Batur Şahin[1*]

[1*] Malatya Turgut Özal University, Faculty of Engineering and Natural Sciences, Departmant of Software Engineering, Malatya, Turkey, (ORCID: 0000-0002-2131-6368), canan.batur@ozal.edu.tr

(First received 11 January 2021 and in final form 6 April 2021)

(**DOI:** 10.31590/ejosat.858720)

ATIF/REFERENCE: Batur Şahin, C. (2021). The Role of Vulnerable Software Metrics on Software Maintainability Prediction. *European Journal of Science and Technology*, (23), 686-696.

Abstract

Software maintainability is among the basic quality features of software engineering. Vulnerability prediction is crucial to protect software maintainability from attacks for cybersecurity. Hence, managing vulnerability in an accurate way is an important phase for the efficient prediction of software maintenance. The existing technologies have achieved many good results in vulnerability detection, but no significant results have been obtained on how effective vulnerability metrics for software maintainability prediction is. As far as we know, this paper is the first study that applies the Deep Learning-based Symbiotic Immune Network Model to develop a software maintainability prediction model using vulnerability software metrics. This study proposes a novel methodology capable of discovering software maintainability metrics in open-source software programs efficiently and accurately. The current study also tries to identify vulnerability metrics frequently utilized in software maintainability. In this paper, five commonly employed open-source projects subjected to attacks, such as Mozilla, Linux Kernel, Xen Hypervisor, glibc, and httpd, are used. In the scope of this research, mentioned five open-source software projects were used as datasets, and they were analyzed with their effect on software maintainability prediction. The analysis of the software metrics was performed, and the descriptive statistics of the software metrics were presented. The current research obtained results of software metrics that accurately predicting software maintenance. Furthermore, the experimental findings confirm the effectiveness of the obtained vulnerability metrics for predicting software maintainability. Our experimental results claim that the proposed Deep Learning-based Symbiotic Immune Network Model enables the prediction of software maintainability to be substantially more effective.

Keywords: Deep-Learning, Immune Network Model, Symbiotic Learning, Software Maintainability, Vulnerability metrics.

Yazılım Sürdürülebilirlik Tahmininde Güvenlik Açığı Yazılım Metriklerinin Rolü

Öz

Yazılım sürdürülebilirliği, yazılım mühendisliğinin temel kalite özellikleri arasındadır. Güvenlik açığı tahmini, yazılım sürdürülebilirliğini siber güvenlik saldırılarına karşı korumak için oldukça önemlidir. Bu nedenle, güvenlik açığının doğru bir şekilde yönetimi, yazılım sürdürülebilirliğinin tahmini için önemli bir aşamadır. Mevcut teknolojiler, güvenlik açığı tespitinde pek çok iyi sonuç elde etmişlerdir, ancak yazılım sürdürülebilirlik tahmini için güvenlik açığı metriklerinin ne kadar etkili olduğu konusunda önemli sonuçlar elde edilmemiştir. Bildiğimiz kadarıyla, bu çalışma, güvenlik açığı yazılım metriklerini kullanarak bir yazılım sürdürülebilirlik tahmin modeli geliştirmek için Derin Öğrenme tabanlı Simbiyotik Bağışıklık Ağı Modelini uygulayan ilk çalışmadır. Bu çalışma, açık kaynaklı yazılım projelerindeki yazılım sürdürülebilirlik metriklerini verimli ve doğru bir şekilde keşfedebilen yeni bir metodoloji önermektedir. Mevcut çalışma aynı zamanda yazılım sürdürülebilirliğinde sıklıkla kullanılan güvenlik açığı metriklerini belirlemeye çalışmaktadır. Bu çalışmada, Mozilla, Linux Kernel, Xen Hypervisor, glibc ve httpd gibi saldırılara maruz kalan, yaygın olarak kullanılan beş açık kaynaklı proje kullanılmıştır. Bu çalışma kapsamında, söz konusu beş açık kaynaklı yazılım projesi veri kümesi olarak kullanılmış ve yazılım sürdürülebilirlik tahminine etkileri ile analiz edilmiştir. Yazılım metriklerinin analizi gerçekleştirilmiş ve yazılım metriklerinin tanımlayıcı istatistikleri sunulmuştur. Mevcut araştırma, yazılım bakımını doğru bir şekilde tahmin eden yazılım metriklerinin sonuçlarını elde etmiştir. Aynı zamanda, deneysel sonuçlar, elde edilen güvenlik açığı metriklerinin yazılım sürdürülebilirliğini tahmin etmede etkinliğini doğrulamaktadır. Deneysel sonuçlar, önerilen Derin Öğrenme tabanlı Simbiyotik Bağışıklık Ağı Modelinin, yazılım sürdürülebilirliği tahmininin önemli ölçüde daha etkili olmasını sağladığını kanıtlamaktadır.

Anahtar Kelimeler: Derin Öğrenme, İmmün Ağ Modeli, Simbiyotik Öğrenme, Yazılım Sürdürülebilirliği, Güvenlik açığı metrikleri.

[*] Corresponding Author: canan.batur@ozal.edu.tr

1. Introduction

It is possible to define software maintainability as the degree of easiness at which the modification of a software system or component can be performed for its correction, improvement, or adaption to its environment. Software maintainability has four major subcategories, such as changeability, analysability, testability, and stability.

There is a high level of correlation between the software metrics and maintainability of software, in other words, it is possible to develop models for predicting maintainability by utilizing software metrics. Furthermore, in object-oriented software, maintainability constitutes an essential quality feature that helps in enhancing the design and coding of software. Object-oriented software metrics predict software maintainability in the best way. Software metrics calculate different software features describing the physical and functional properties of a process, component, or project. Moreover, software metrics help developers discover and fix mistakes. Vulnerability discovery metrics have considerable potential to shed light on the software maintainability failures that may have caused introducing vulnerabilities. In the present study, software vulnerability metrics are utilized as an indicator of the maintainability prediction model.

Vulnerability detection play a significant part in software security and quality [1]. This paper discussed the subject of detecting the correlation of the most relevant vulnerability metrics in software maintainability by utilizing deep learning. Object-oriented metrics were analyzed, and their effect on open-source software maintainability was also investigated. This study's goal is to obtain a relationship between vulnerability metrics and software maintainability. The relationship is determined by a deep learning-based symbiotic immune network model. Generally, Deep learning applied based on the neural network architecture [2]. In this paper, the learning mechanism from vulnerability metrics is demonstrated and modeled for predicting software maintainability. Hence, a symbiotic mechanism is employed.

There is a number of software maintainability prediction techniques that are combined with popular learning-based approaches. In [3], deep learning was applied for predicting software maintainability metrics on many datasets. The said study obtained findings in the form of metrics that might be utilized for predicting software maintenance, and the suggested deep learning model was superior to all other methods analyzed. In [4], a suitable model was developed using a hybrid neural network to predict the maintainability of object-oriented software by utilizing class-level metrics. The findings demonstrated that the model developed by the suggested hybrid approach yielded better results in comparison with the related works. In [5], which used deep learning to detect vulnerabilities at the slice level for the first time, it was indicated that other studies on the usage of deep learning to detect vulnerabilities were at a coarser granularity (e.g., function level). VulDeePecker shows how feasible the use of deep learning for vulnerability detection is. In [6], the feasibility and advantages of implementing deep learning techniques to analyze and detect software vulnerabilities were investigated. Furthermore, this paper addressed different vulnerability databases/resources and a number of the recent successful deep learning applications in the prediction of vulnerabilities in the software. In [7], a model was created for developing stable associative memory, which can solve robustness and optimization tasks. The LSTM was utilized to better understand the mechanisms containing the "remember" attribute of the immunological behavior of the immune response.

The principal contributions of the current study are given below.

- This paper is the first model that predicts software maintainability using vulnerable software metrics with the deep learning-based symbiotic immune network model. This makes the proposed methodology an original approach to effectively detect and analyze software maintainability prediction.
- This paper introduces a novel framework trained by Long Short-Term-Memory (LSTM) and Gated Recurrent Unit (GRU) Recurrent Neural Networks (RNNs) for learning deep correlated vulnerable software metrics to detect software maintainability. The selected object-oriented metrics are utilized in predicting software maintainability.
- In the current research, we accept determining the correlated software metrics from the symbiotic immune network model as a combinatorial optimization problem. Therefore, we propose a novel methodology called the Symbiotic Immune Network.

The remaining part of this paper is organized in the following way. The preliminaries are explained in Part 2. Part 3 contains a description of the methods. The proposed method is presented in Part 4. The experimental results and discussion are shown in Part 5. The conclusion and future studies are presented in Part 6.

2. Preliminaries

2.1. Vulnerability Metrics for Software Maintainability

It is crucial and difficult to maintain software security during the whole software life cycle. Vulnerabilities of software systems can lead to various problems, such as deadlock, information loss, or system failure. Nowadays, it is becoming more challenging to manage software security due to its increasing complexity and diversity. During the establishment of a novel software system, software maintainability should be taken into account together with secure software design principles and secure software development life cycles. Researchers have recently utilized vulnerability prediction approaches based on software metrics for the detection of vulnerable codes early for software maintainability.

2.2. Deep Learning for Software Maintainability Prediction Models

Software maintainability prediction models have been studied to help organizations with utilizing costs, allocating resources, and acquiring an accurate management plan and efficient maintenance process. Nevertheless, it is difficult to predict software maintainability, and accurate prediction models are needed for this. Deep learning approaches can discover latent features that a human expert may never think of including, which leads to the significant expansion of the feature search space. This means understanding the vulnerability metrics for maintainability prediction models so that deep learning-based detection systems can learn from these metrics.

2.3. Immune Network Theory

Being a metaphor for different aspects of the natural immune system, the artificial immune network represents a network of connected recognition cells that learn using feedback mechanisms. The fundamental idea of immune networks is that, in the case of the recognition of invasive antigens by antibodies, various antibodies make up a dynamic network by interacting between themselves. The immune system represents an interacting network of lymphocytes and molecules with variable (V) regions. Thus, the immune system is regarded as a network having the components that are connected by V-V interactions. The structure of the immune network represented in figure 1.

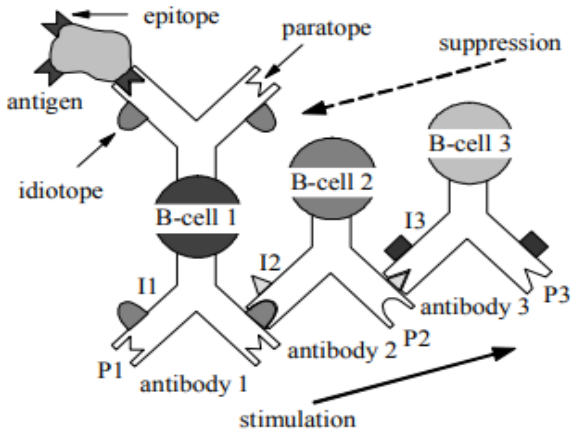

Figure 1: Structure of immune networks [8].

3. Methods

3.1. Deep-Learning Based Classifiers

Deep learning (DL) represents a branch of machine learning models. Its ability to extract hierarchical representations from input data as a result of the establishment of deep neural networks having multiple layers of nonlinear transformations characterizes deep learning.

3.2. Recurrent Neural Network

Recurrent neural networks (RNNs) can memorize arbitrary length sequences of input patterns by establishing relations between units. The transition function at every time step (t) takes the current time information, which is denoted as Xt, and h(t-1) is the previously hidden output. The updating of the current hidden output is performed using Equation (1):

$$ht = H(Xt + h(t-1)) \quad (1)$$

In Equation (1), H refers to a nonlinear and differentiable transformation function. When the complete sequence is processed, the hidden output at the final time step, in other words, ht, may be considered as a vector of sequential data. The addition of the supervised learning layer on top is performed with the aim of mapping the acquired representation ht to targets, and it is possible to train the model via backpropagation through time.

3.3. Long-Short-Term-Memory (LSTM)

LSTM networks represent one of the most effective solutions to a sequence of prediction problems because of the recognizing patterns in data sequences. Since LSTM networks have a particular type of memory, they can selectively remember patterns for a long time. They represent quite a reasonable approach to predict the period with the unknown long delays that occur between important events. The LSTM memory block's structure is composed of three gates and a self-recurrent connection.

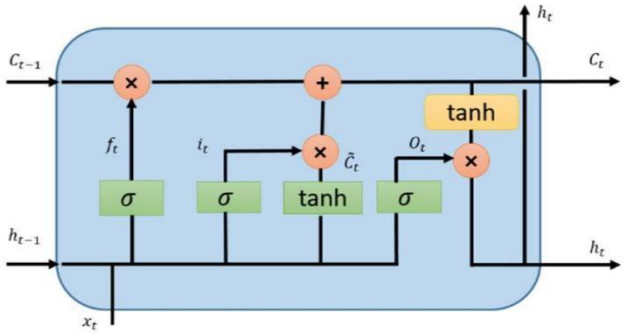

Figure 2: Architecture of the LSTM Recurrent Neural Network [9].

C_t refers to the memory amount of LSTM unit at time t. The output of the LSTM at time t is represented by the h_t. LSTM unit, σ_t denotes the output gate managing the memory content exposure. σ denotes the sigmoid function, $\hat{C}_t$ refers to a novel memory content of the memory unit, which is updated by partly forgetting the current memory and adding the novel memory content to C_t.

The current memory forgetting gate is modulated by f_t. The addition degree of the new memory content to the memory cell is modulated by an input gate it.

3.4. Gated Recurrent Unit (GRU)

GRU represents an improved version of standard recurrent neural networks (RNNs) [12]. GRU networks have two gates: a reset gate (r), which performs the adjustment of incorporating novel input with the previous memory, and an update gate (z), which performs the control of preserving the previous memory.

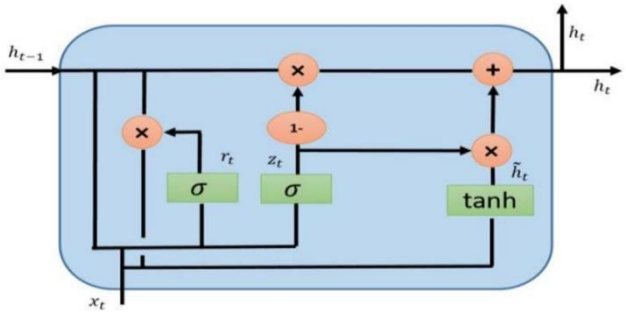

Figure 3: Architecture of the GRU Recurrent Neural Network [9].

The update gate helps the model determine the amount of the previous information (from past time steps). Identically, it is applicable to h_{t-1}, which holds the information for the past *t-1* units. A sigmoid activation function is employed to squash the outcome between 0 and 1. The reset gate is utilized so that the model can make a decision on the past information amount necessary to forget. The update gate determines what it is required to collect from the current memory content $\hat{h}_t$ and what from the past steps h_{t-1}, final memory at the present time step.

4. Proposed Method

The proposed model is a computational modeling paradigm that depends on the immense detection and prediction capability of immune neural networks. We construct a deep learning-based symbiotic immune memory network model that discovers software maintainability from vulnerability metrics with characteristics that are more prone to software security. This makes this methodology a novel approach toward the effective detection and analysis of software maintainability. Thus, the deep learning-based symbiotic immune network model can adaptively learn useful maintainability metrics.

4.1. The Proposed Deep-Learning Based Symbiotic Immune Network Model

In the proposed framework, the structure of the immune network is utilized to identify the metrics of a software maintainability candidate solution. In the current study, we suggest a new cell interaction model in a symbiotic manner in which antibodies interact with cells. We investigate how to utilize a deep neural network (DNN) and immune network to predict software maintainability based on vulnerability software metrics. The symbiotic immune network displays an Eigen-behavior resulting from cell-cell interactions of antibodies in (V) regions within the immune system as a co-evolution system. The immune network-based symbiotic mechanism plays a major role in an interacting network of lymphocytes and molecules having variable (V) regions. The co-evolution of the populations of heterogeneous antibodies affect the formation of idiotypic networks. Thus, networks are updated in dynamic progress by integrating the novel memorized auto-reactive cells in the network. Therefore, the captured long context correlations, in which the dependent software metrics are in V regions, are obtained and used to predict the software maintainability.

$$f(Abi,Agj)=1/(1+\|Abi,Abj\|) \quad (2)$$

Each data was assumed to be denoted as follows:

x = {x1, x2, . . ., xN}, where N refers to the length of the training data (vulnerable software metrics).

The fitness function in Equation (2) is used to reveal the quality of every interacting antibody in novel deep-symbiotic memory. We use the Euclidean distance of two software metric (SM) vectors to measure their similarity. Each SM serves as an antibody.

In the proposed framework, an enhanced symbiotic immune network based on deep learning methods is proposed. Each LSTM and GRU recurrent neural network (RNN) model uses its functions in the graph as a transformation/aggregation function. For each deep learning model, neurons aggregate information from their neighbors using a symbiotic immune neural network.

The meaning of the symbols in the pseudocode of proposed model is as follows: *Ab* refers to the antibodies in repertoire, *S* refers to the similarity matrix between every pair of antibody, *C** denotes the vector that contains the affinity between each element Abj, *dj* denotes the vector that contains the affinity between each element from the set *C** with other Ab, ζ refers to the percentage of the mature antibodies that should be chosen, *Mj* is the memory clone for antibody *Abj* (that remains from the clonal suppression process), *M*j* is the resultant clonal memory, *σd* is the natural death threshold, *σs* refers to the suppression threshold, and *σcut* refers to the cutting threshold. The Euclidean distance between the antibodies, capable of forming the affinity matrix, expresses Dk,j. In Immune-network-model, matrix *Ab* antibody pool is presented as software metrics to the immune network, and matrix *S* identifies the connections among more prone vulnerable antibodies.

For the detection of code-clones, software metrics are utilized with the aim of pooling the nodes into a network-level vector representation for every immune network in a separate manner. Then, the quantification of the affinity between the interactions of an antibody and other antibodies is performed by the measurement of the similarity degree (affinity). The affinity is employed for the detection of behavioral equivalence between SM vectors, which is then generalized to the vulnerability.

In the proposed framework, an enhanced immune network methodology based on DNN methods is proposed. Each LSTM and GRU recurrent neural network (RNN) model uses its functions in the graph as a transformation/aggregation function. For each RNN model, nodes aggregate information from their neighbors using an immune neural network.

$$SYMB\_M = finitial\left(h_i^t, h_j^t\right) \quad \forall i, j \in E \quad (3)$$

$$SYMB\_M^* = faggregate(SYMB\_Mj \rightarrow i \mid \forall i, j) \quad (4)$$

$$h_j^{(t+1)} = fupdate\left(h_j^t, SYMB\_M^*\right) \quad (5)$$

Procedure: **Deep-Symbiotic Immune Network Model**

Input: Set of vectors of vulnerable code-metrics
Output: The list of software metrics more prone to have security vulnerabilities

Step 1: {*Initialize Antibody Pool* (Software metrics)}
Step 2: [Train] {1...N} (Input Size)
Step 3: **For** each iteration, **do**:
Step 4: **For** each antibody Abj, j = 1,...,N, (Abj ∈ Ab), **do**:
Step 5: Determine fitness (affinity) of each antibodies in population P
Step 6: From C*, re-select ζ% of the antibodies with highest dk,j and put them into a matrix Mj of symbiotic memory S;
Step 7: SYMB_M=*CreateDNNSymbioticMemory*(*Mj, t,State_id*)
Step 8: Co-evaluation of interacting antibodies of SYMB_M in dynamic V region
Step 8: Apoptosis: eliminate all the memory clones from SYM_M whose affinity Dk,j > σd:
Step 9: Determine the affinity Si,k among the DNNclonalmemory. Si,k=|| SYMB_M j,i − SYMB_M j,k ||, ∀i ,k
Step 10: Clonal suppression: eliminate those DNN memory clones whose si,k < σs:
Step 11: Concatenate the total antibody memory matrix with the resultant DNN clonal memory SYMB_Mj * for Abj: Ab{m} ← [Ab{m}; SYMB_M j *]
Step 12: DNNM* ←*UpdateDNNClonalMemory* (SYMB_M j, *t,State_id*)
Step 13: *State_id* ← *State_id+1*
Step 14: Determine the affinity among all the DNN clonal memory SYMB_M * antibodies from Ab{m}: Si,k=|| Ab^i{m} − Ab^k{m} ||, ∀i ,k
Step 15: Network suppression: eliminate all the antibodies such that Si,k < σs:
Step 16: **END For**
Step 17: Build the total antibody matrix Ab← [Ab{m}]
Step 18: Add list of more prone security metrics
Step 19: **END For**

Figure 4: Pseudocode of proposed model.

Where $f_{initial}$ represents the initial function, whereas f_{update} represents the antibody neurons update function. $f_{aggregate}$ denotes an aggregation function, use as a direct sum. Equations (3) and (4) can be regarded as aggregators in which every antibody node collects information from its neighbors. Equation (5) represents an updater, updating the hidden state of all nodes. The structure of Sub-Deep-Symbiotic Immune Network and sub-network representation for correlated metrics presented in Figure 5-6, respectively. Figure 7 shows the schema of the proposed Deep-learning based symbiotic immune network model.

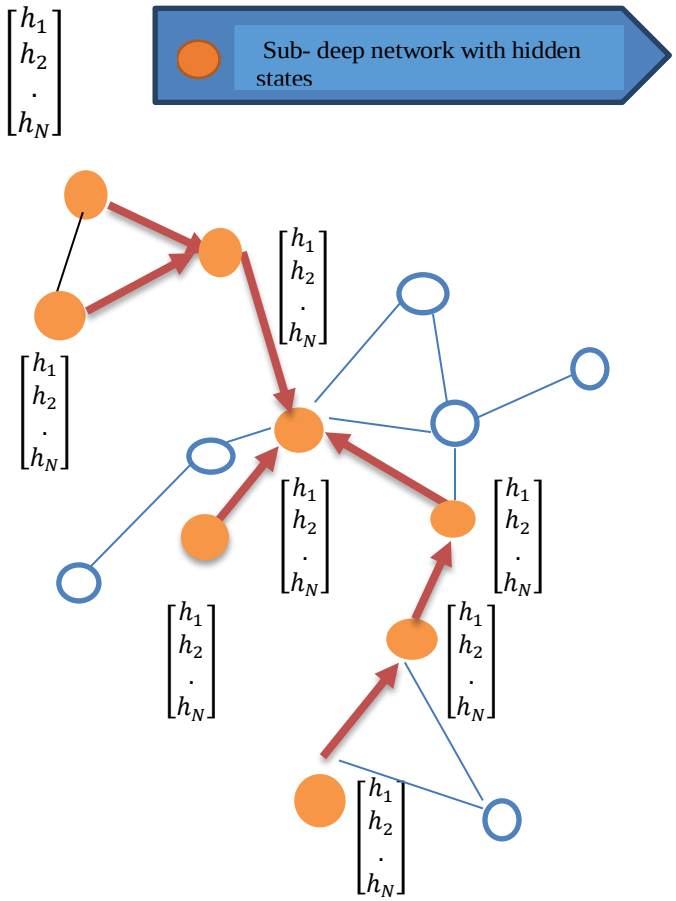

Figure 5: Structure of sub deep-symbiotic immune network.

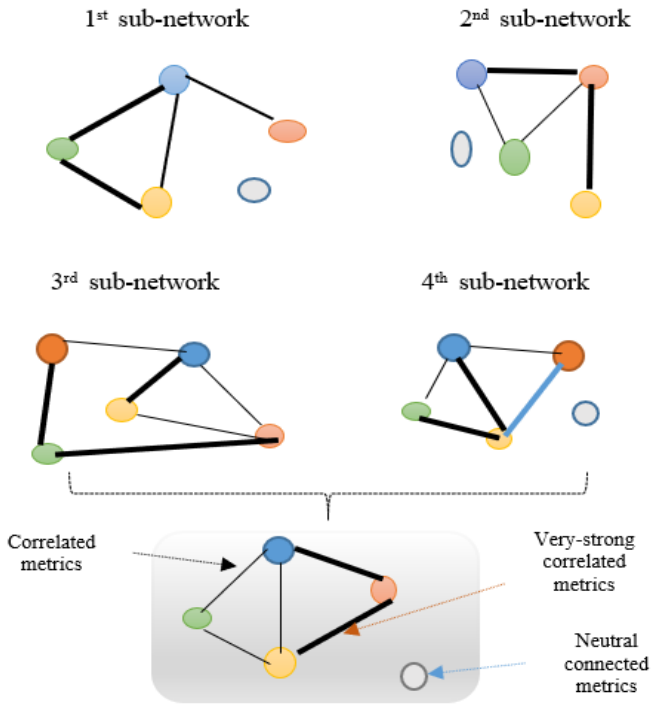

Figure 6: Sub-network representation for correlated metrics.

5. Experimental Results and Discussion

In the current part, experiments are conducted to assess the suggested methods. For the evaluation, we consider five commonly employed open-source projects subjected to attacks, such as Mozilla, Linux Kernel, Xen Hypervisor, glibc, and httpd (https://eden.dei.uc.pt/~nmsa/metrics-dataset/).

In the present research, we introduce an approach toward utilizing vulnerability discovery metrics to ensure insightful feedback for software maintainability. We identified a lot of potential metric-set combinations to take a decision on the correlation of vulnerable metric sets for software maintainability. The findings demonstrate that it is possible to use the dataset for the purpose of distinguishing which metrics are more prone to detect security vulnerabilities. The best vulnerable software metric sets for the suggested LSTM-symbiotic network and GRU-symbiotic network are shown in Table 1 and Table 2, respectively. The random forest was employed as a classifier. The findings showed that using particular software metrics, including Max cyclomatic for the LSTM-symbiotic network and percent lack of cohesion for the GRU-symbiotic network, it was possible to achieve a high accuracy rate above 98% in software maintainability metrics prediction.

Figure 8-17 plots the Root Mean Square Error (RMSE)-measure of Deep Learning based Symbiotic Immune Network and with respect to the six-software metrics with different number of hidden layers. We observe that the generally RMSE-measure of the six-software metrics achieve good results at 0-5 layers, and the RMSE-measure of most of these 6-software metrics increases when the number of layers is great. To empirically test the effect of the number of hidden layers, we assessed and compared the model results for each project based on six software metrics. The RMSE was computed to analyze the impact of which software metrics useful for robust software maintainability prediction. As shown in Figures 8-12, the proposed LSTM-Symbiotic Immune Network achived more successful results respectively for encapsulation, inheritance, coupling, polymorphism, complexity and size metrics in detecting httpd, glibc, mozilla, linux kernel and xen hypervisor projects, respectively. As the results are depicted in figures 13-17, the proposed GRU- LSTM-Symbiotic Immune Network outperforms encapsulation, size, inheritance, complexity, coupling and polymorphism metrics for httpd, glibc, Linux kernel, Mozilla and Xen Hypervisor projects respectively, since it produces the lowest change in RMSE.

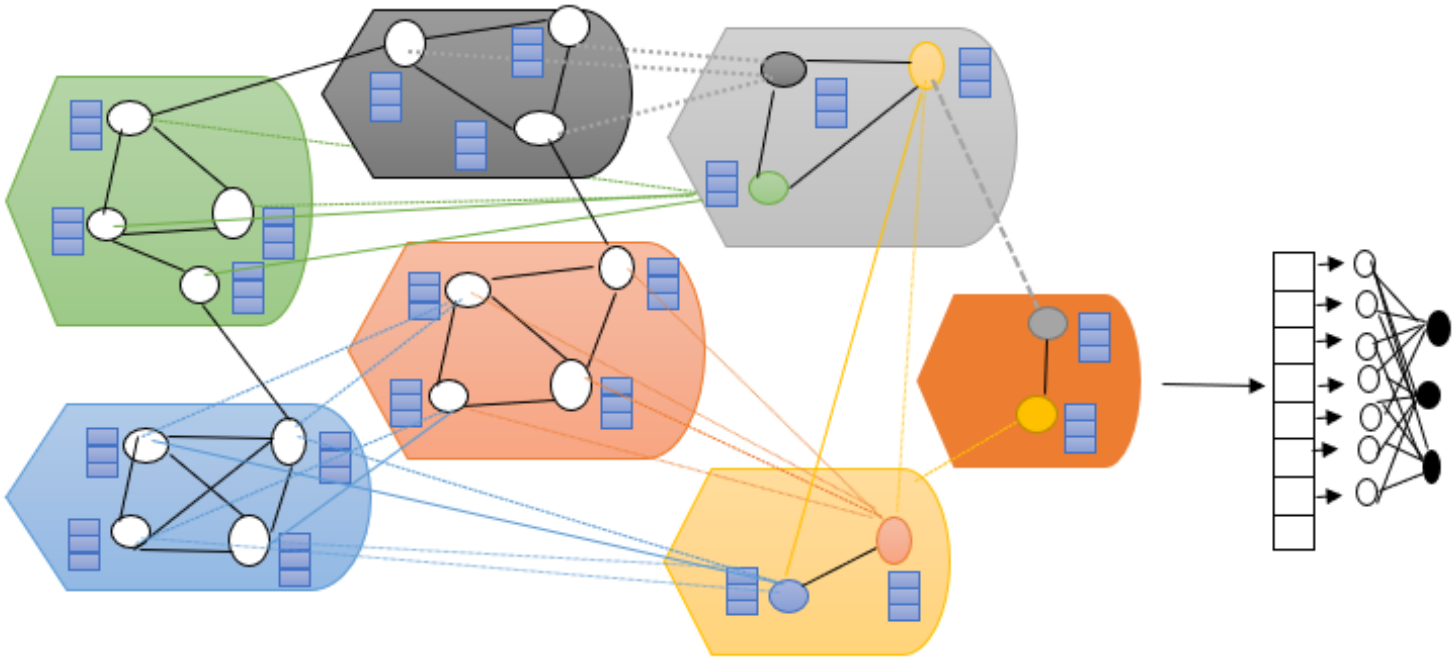

Figure 7: The schema of the proposed Deep-learning-based symbiotic Immune network model.

Table 1. Accuracy of Best Vulnerable metrics for software maintainability based on LSTM-Symbiotic Immune Network

Metric-Set	**Linux Kernel (%)**	**Mozilla (%)**	**Xen Hypervisor (%)**	**Httpd (%)**	**Glibc (%)**
Inheritance tree	90.4	96.7	92.1	97.8	91.6
Max Nesting	91.7	89.5	97.4	96.2	93.5
CountLine	95.3	93.6	95.5	95.8	92.5
Max Cyclomatic	96.8	94.7	94.6	**98.6**	92.7
Count Path	92.4	94.1	95.7	94.5	94.3
Percent Lack of Cohesion	95.8	93.8	94	97.2	92.1

Table 2. Accuracy of Best Vulnerable metrics for software maintainability based on GRU-Symbiotic Immune Network

Metric-Set	Linux Kernel (%)	Mozilla (%)	Xen Hypervisor (%)	Httpd (%)	Glibc (%)
Inheritance tree	91.2	95.8	93.6	98.4	92
Max Nesting	93.8	95.9	94.9	97.3	95.3
CountLine	94.1	97.5	96.3	95.9	93.7
Max Cyclomatic	94.6	96.3	96.2	98.1	94.1
Count Path	95.9	95.2	93.6	97.3	96.8
Percent Lack of Cohesion	93.8	97.2	96.3	**98.7**	93.2

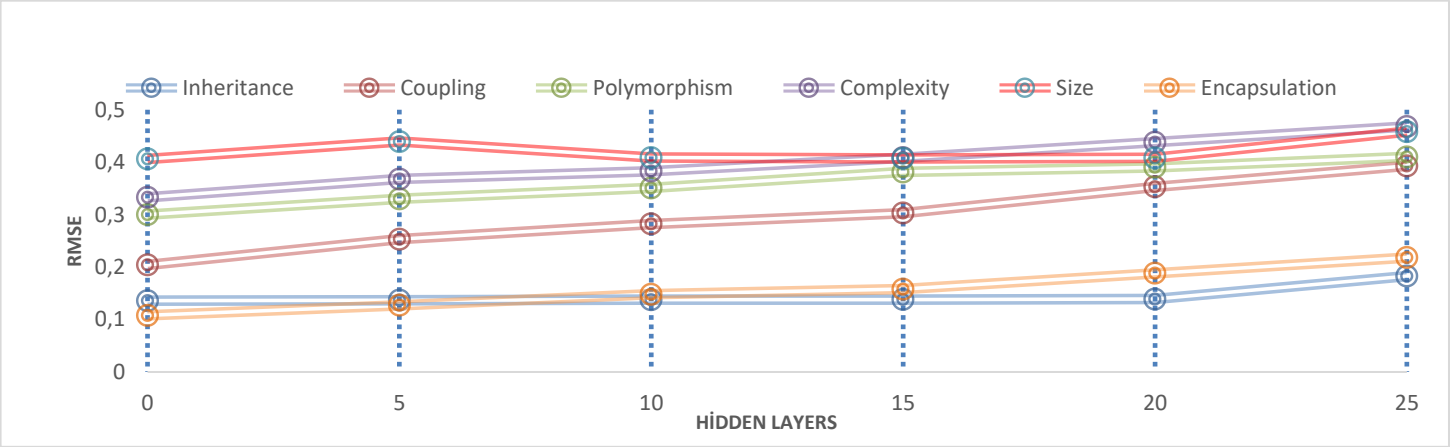

Figure 8. The relationship between number of hidden layers and root mean square error of different software metric categories for LSTM-Symbiotic Immune Network – Linux Kernel Project

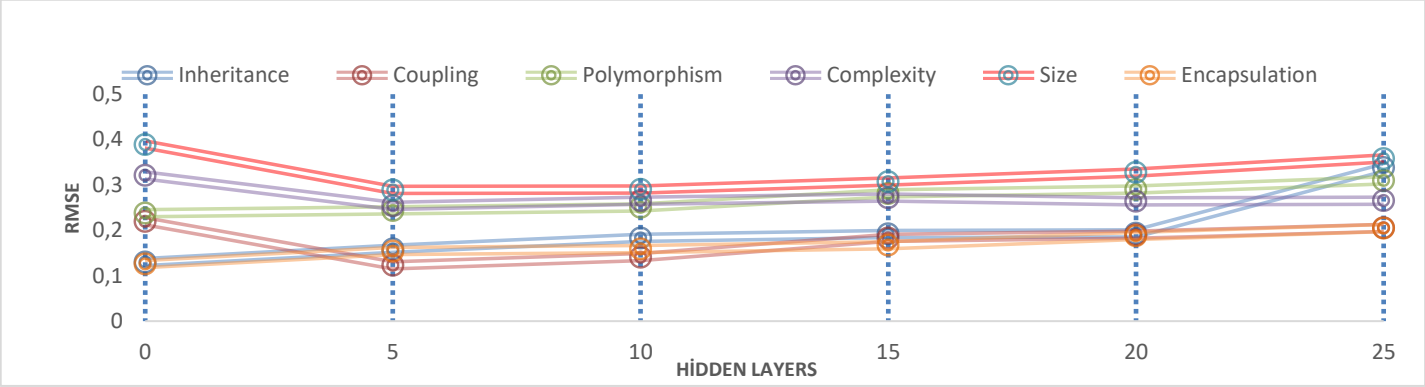

Figure 9. The relationship between number of hidden layers and root mean square error of different software metric categories for LSTM- Symbiotic Immune Network – Mozilla Project

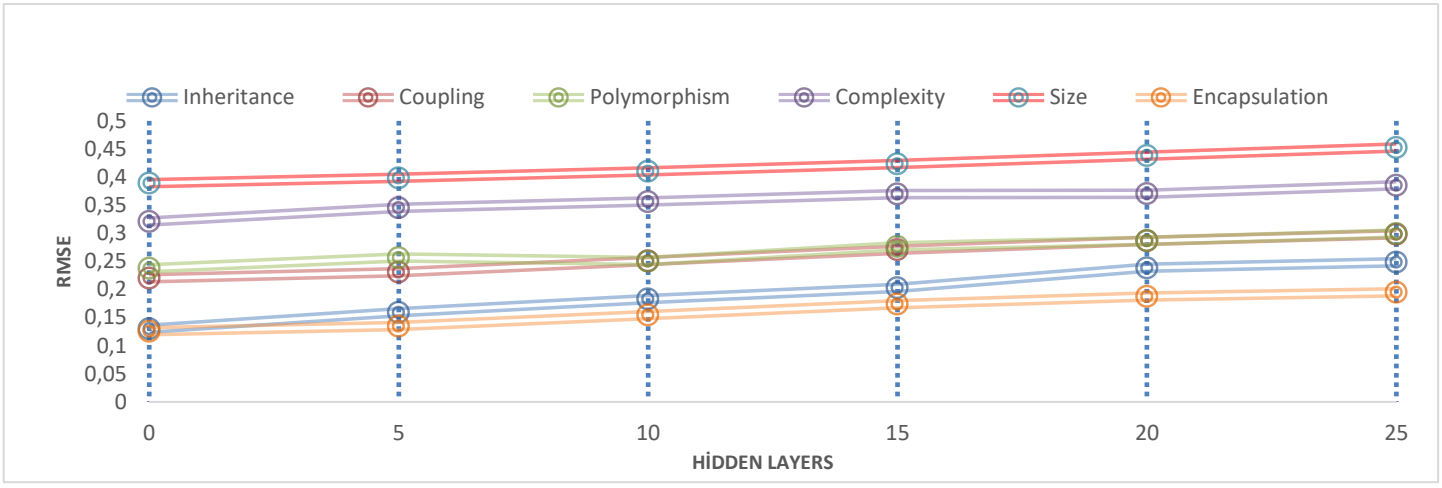

Figure 10. The relationship between number of hidden layers and root mean square error of different software metric categories for LSTM- Symbiotic Immune Network – Xen Hypervisor project

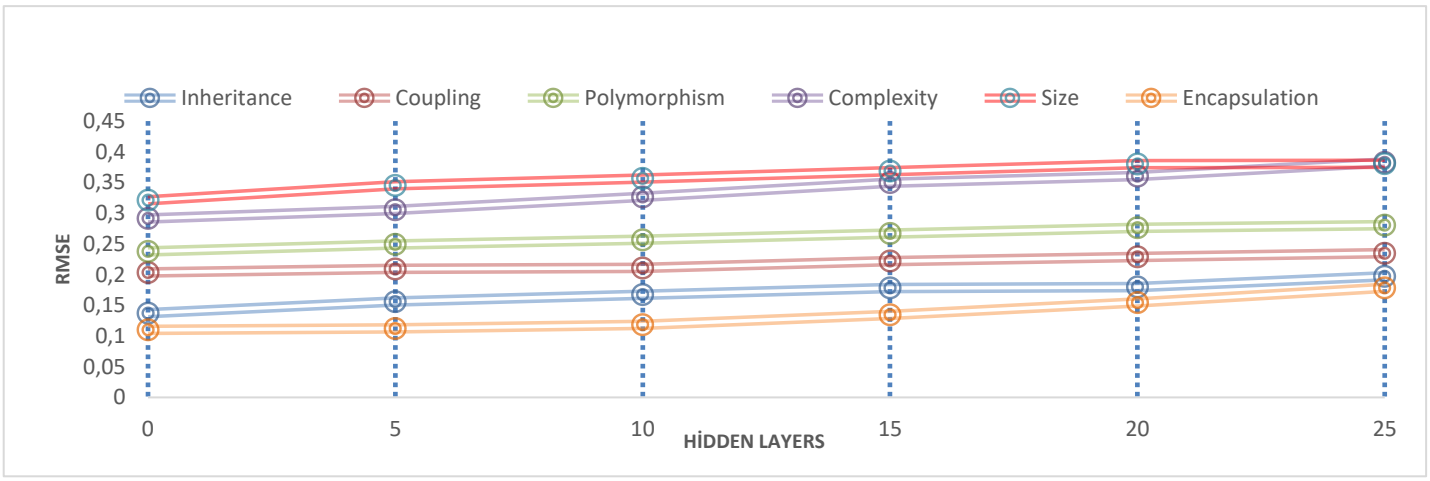

Figure 11. The relationship between number of hidden layers and root mean square error of different software metric categories for LSTM- Symbiotic Immune Network – Httpd Project

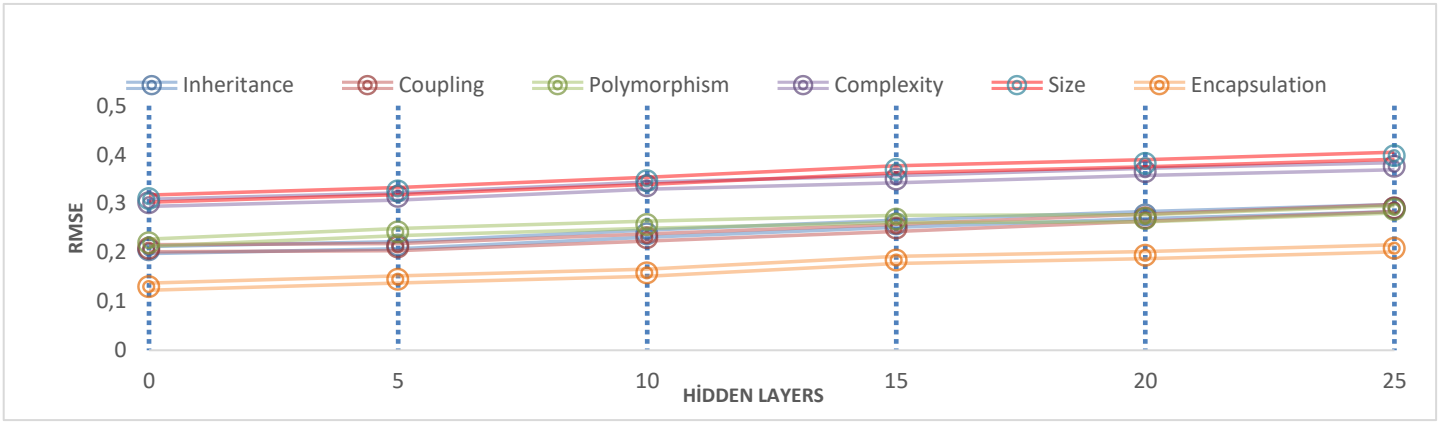

Figure 12. The relationship between number of hidden layers and root mean square error of different software metric categories for LSTM- Symbiotic Immune Network – Glibc Project

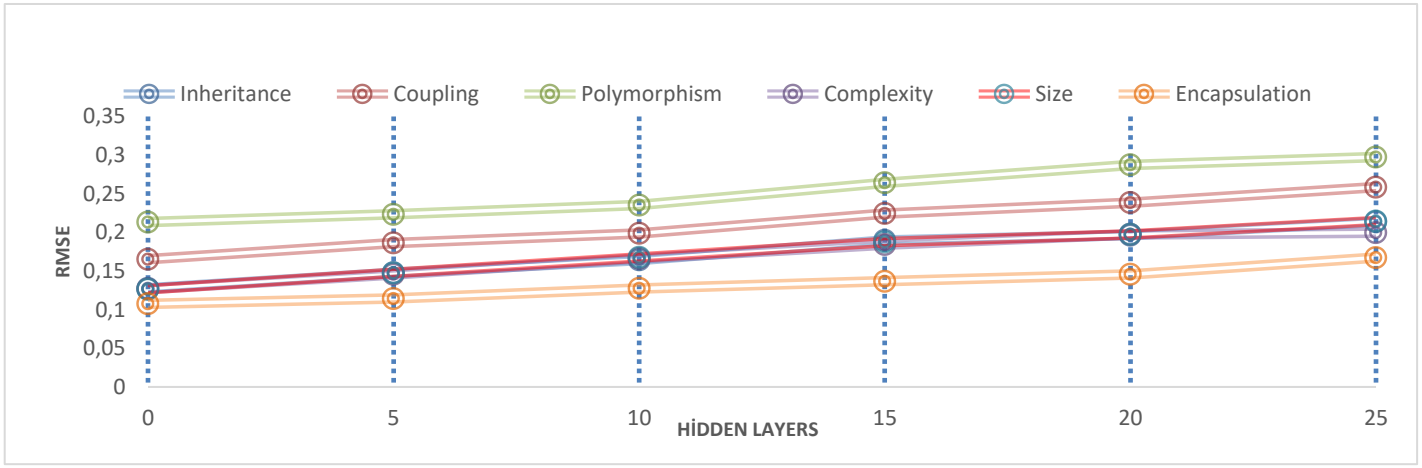

Figure 13. The relationship between number of hidden layers and root mean square error of different software metric categories for GRU- Symbiotic Immune Network – Linux-Kernel project

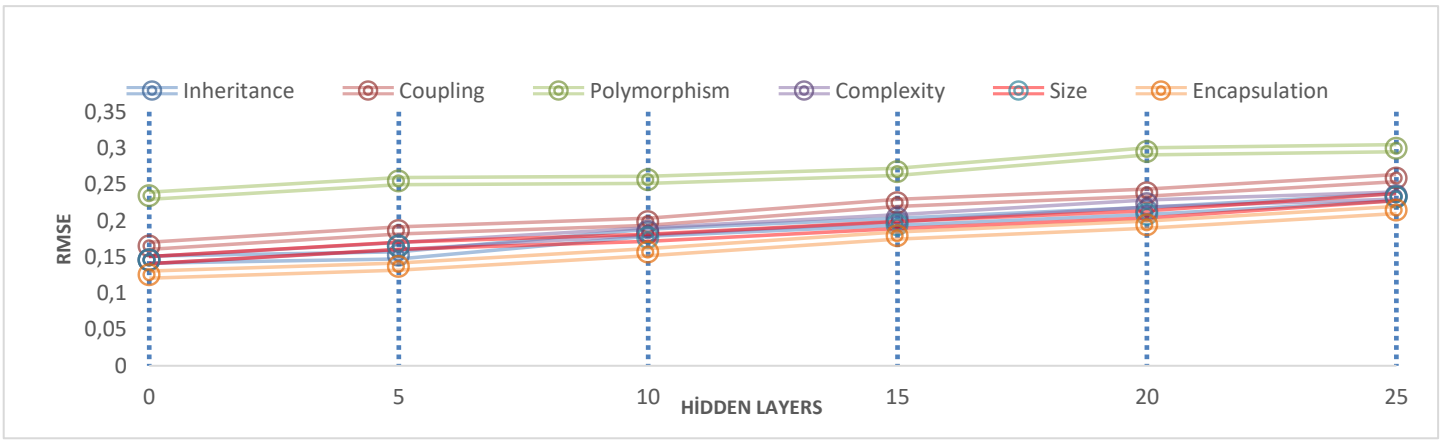

Figure 14. The relationship between number of hidden layers and root mean square error of different software metric categories for GRU- Symbiotic Immune Network – Mozilla Project

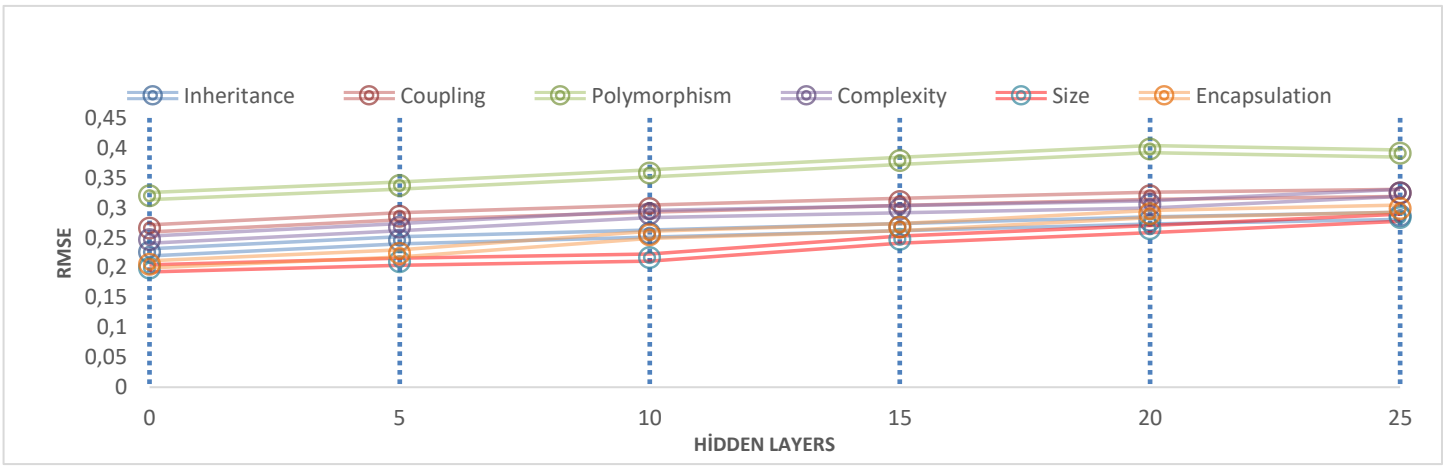

Figure 15. The relationship between number of hidden layers and root mean square error of different software metric categories for GRU- Symbiotic Immune Network – Xen Hypervisor Project

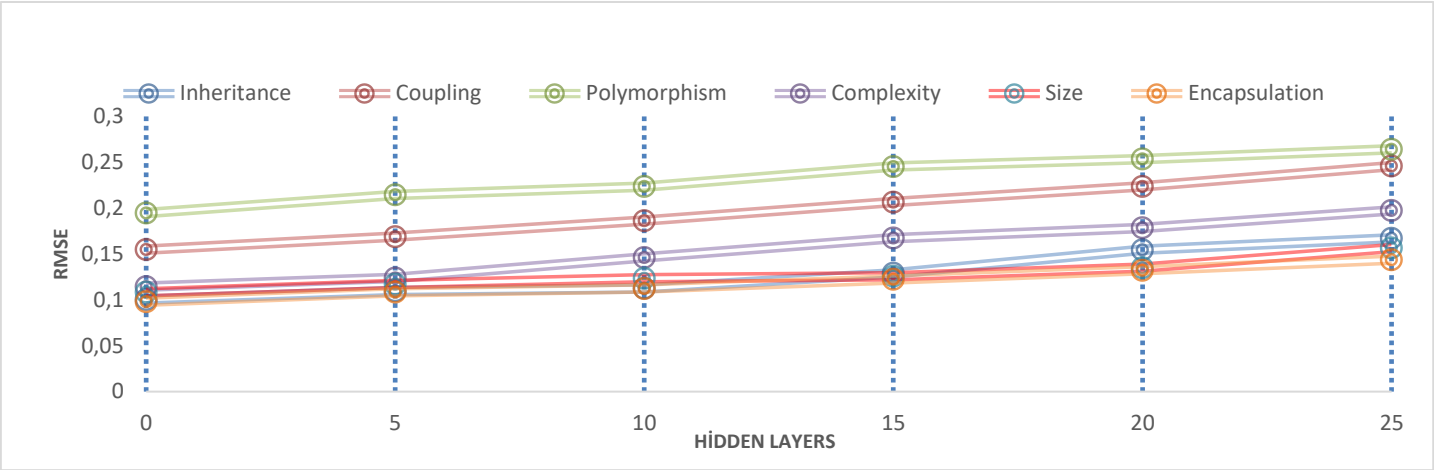

Figure 16. The relationship between number of hidden layers and root mean square error of different software metric categories for GRU- Symbiotic Immune Network – Httpd project

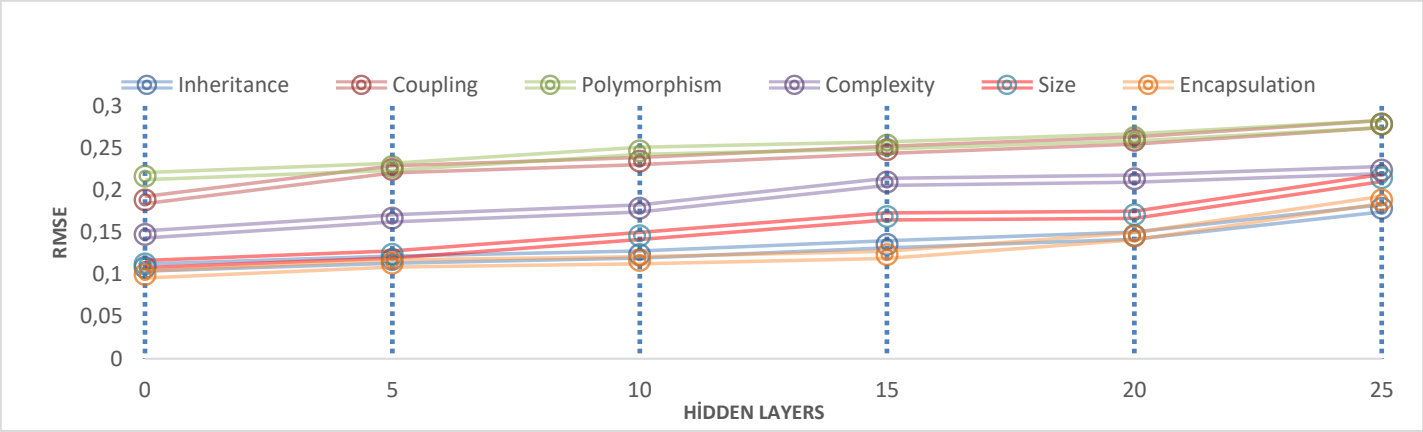

Figure 17. The relationship between number of hidden layers and root mean square error of different software metric categories for GRU- Symbiotic Immune Network – Glibc Project.

6. Conclusion

Software maintenance represents a costly activity consuming a significant part of the total project's cost [14]. Because it is very challenging to track the maintenance behavior of software, it becomes difficult to predict cost and the risk associated. Using maintainability, it is possible to predict what changes or failures can emerge in software following its deployment. In the present study, we suggest a novel methodology for software maintainability prediction and implement the said method on vulnerability metrics. The proposed method was used for identifying the important vulnerable software metrics that help enhance the accuracy of software maintainability. The current research suggests and builds a new framework symbiotic immune network on the basis of deep learning in order to improve robustness to predict software maintainability.

The paper demonstrated that utilizing software metrics with the symbiotic immune neural network was a good method of software maintainability analysis and prediction. In conclusion, the findings demonstrate that vulnerable metrics have major effects on software maintainability, and it is possible that they will have other vulnerabilities hereafter. The software maintainability in the symbiotic neural network will be discovered by supplementary improvement approaches in the future.

5. Acknowledge

This article does not contain any studies with human participants performed by any of the authors.

References

[1] Batur Şahin C., Batur Dinler Ö., Abuagilah L. (2021). Prediction of software vulnerability-based deep symbiotic genetic algorithms: Phenotyping of dominant-features, Applied Intelligence, doi: 10.1007/s10489-021-02324-3.

[2] Batur Dinler, Ö , Batur Şahin, C . (2021). Prediction of Phishing Web Sites with Deep Learning Using WEKA Environment . European Journal of Technique ,35-41 . DOI: 10.31590/ejosat.901465

[3]Jha S. et. al., (2020). Deep Learning Approach for Software Maintainability Metrics Prediction, IEEE Access, doi: 10.1109/ACCESS.2019.2913349.

[4] Kumar L., Lal S., and Murthy L.B., (2019). Estimation of maintainability parameters for object-oriented software using hybrid neural network and class level metrics, Int J Syst Assur Eng Manag 10, https://doi.org/10.1007/s13198-019-00853-2, 1234–1264.

[5] Li Z., et al., (2019). VulDeePecker: A Deep Learning-Based System for Vulnerability Detection, Cryptography and Security, Doi: 10.14722/ndss.2018.23158.

[6] Singh S.K., Chaturvedi A., (2020). Applying Deep Learning for Discovery and Analysis of Software Vulnerabilities: A Brief Survey, Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol 1154. Springer, Singapore. https://doi.org/10.1007/978-981-15-4032-5_59.

[7] Şahín C. B., and Dírí B., (2019). Robust Feature Selection with LSTM Recurrent Neural Networks for Artificial Immune Recognition System, in IEEE Access, vol. 7, pp. 24165-24178, doi: 10.1109/ACCESS.2019.2900118.

[8] Tsankova D., et al., (2007). Modeling Cancer Outcome Prediction by aiNet: Discrete Artificial Immune Network, Proceedings of the 15th Mediterranean Conference on Control&Automation, Jully 27-29, Athens, Greece.

[9] Alom M. Z., Taha T. M., et al., (2019). A state-of-the-art survey on deep learning theory and architectures. Electronics, 8, 292; doi:10.3390/electronics8030292.

[10] Dai H., and Li C., (2009). Immune Network Theory Based Artificial Immune System and Its Application, Second International Conference on Intelligent Networks and Intelligent Systems.

[11] Alsolai H., Roper M., (2020). A systematic literature review of machine learning techniques for software maintainability prediction. Information and Software Technology, doi: 10.1016/j.infsof.2019.106214.

[12] Ardito L., Coppola R., Barbato L., and Verga D., (2020). A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review, https://doi.org/10.1155/2020/8840389.

[13] Munaiah N,. and Meneely A., (2019). Data-Driven Insights from Vulnerability Discovery Metrics, IEEE/ACM Joint 4th International Workshop on Rapid Continuous Software Engineering and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution (RCoSE/DDrEE), doi: 10.1109/RCoSE/DDrEE.2019.00008.

[14] Kalıpsız, O , Cihan, P . (2016). Öğrenci Proje Anketlerini Sınıflandırmada En İyi Algoritmanın Belirlenmesi. Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi, 8 (1), 41-49.

[15] Mishra S., and Sharma A., (2015). Maintainability prediction of object-oriented software by using adaptive network based fuzzy system technique. International Journal of Computer Applications, 119(9): 1154-1168.

[16] Li Z., Zou D., Xu S., Jin H., Zhu Y., and Chen Z., (2018). SySeVR: A framework for using deep learning to detect software vulnerabilities. ArXiv:1807.06756. [Online]. Available: https://arxiv.org/abs/1807.06756.

[17] Liu S., et. al., (2020). CD-VulD: Cross-Domain Vulnerability Discovery based on Deep Domain Adaptation, IEEE Transactions on Dependable and Secure Computing, Doi:10.1109/TDSC.2020.2984505. pp: (99): 1-1.

[18] Li Y., Tarlow D., Brockschmidt M., and Zemel R. S., (2015). Gated graph sequence neural networks. CoRR, abs/1511.05493.

[19] Zagane M., and Abdi M. K., (2019). Evaluating and comparing size, complexity and coupling metrics as Web applications vulnerabilities predictors, Int. J. Inf. Technol. Comput. Sci., vol. 11, no. 7, pp. 35–42, Jul.