Neural Computing and Applications (2021) 33:14049–14067
https://doi.org/10.1007/s00521-021-06047-x

ORIGINAL ARTICLE

A novel deep learning-based feature selection model for improving the static analysis of vulnerability detection

Canan Batur Şahin[1] · Laith Abualigah[2]

Received: 21 October 2020 / Accepted: 15 April 2021 / Published online: 1 May 2021

Abstract
The automatic detection of software vulnerabilities is considered a complex and common research problem. It is possible to detect several security vulnerabilities using static analysis (SA) tools, but comparatively high false-positive rates are observed in this case. Existing solutions to this problem depend on human experts to identify functionality, and as a result, several vulnerabilities are often overlooked. This paper introduces a novel approach for effectively and reliably finding vulnerabilities in open-source software programs. In this paper, we are motivated to examine the potential of the clonal selection theory. A novel deep learning-based vulnerability detection model is proposed to define features using the clustering theory of the clonal selection algorithm. To our knowledge, this is the first time we have used deep-learned long-lived team-hacker features to process memories of sequential features and mapping from the entire history of previous inputs to target vectors in theory. With an immune-based feature selection model, the proposed approach aimed to improve static analyses' detection abilities. A real-world SA dataset is used based on three open-source PHP applications. Comparisons are conducted based on using a classification model for all features to measure the proposed feature selection methods' classification improvement. The results demonstrated that the proposed method got significant enhancements, which occurred in the classification accuracy also in the true positive rate.

Keywords Software vulnerability prediction · Deep learning · Feature selection · Immune systems

1 Introduction

Software vulnerabilities constitute an increasingly important security risk for software systems, which utilizes attacking and damaging the system [1]. A software vulnerability is possibly defined as a weakness that attackers can use to access or modify confidential characters' data in a software-intensive system [2]. 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. When establishing a novel software system, security techniques should be taken into account with secure software design principles and secure software development life cycles [3].

Vulnerability detection at an early stage assists businesses with fixing vulnerabilities before the deployment stage, which brings the testing costs to a minimum and maintains their reputation in the market. Vulnerability prediction involves determining the possibility of a software component being vulnerable or not. The detection of vulnerabilities plays an essential role in development and software maintenance, so a vulnerability prediction model's accuracy is necessary [4]. Despite many tools and techniques for software security, software security vulnerabilities still occur very frequently, and increasing the effectiveness of prediction techniques is still a challenging research question. Due to the limitations of the available methods, tools, and techniques, researchers have begun to develop novel prediction models, named software vulnerability prediction (SVP). SVP models are created primarily

✉ Canan Batur Şahin
canan.batur@ozal.edu.tr

Laith Abualigah
Laythdyabat@aau.edu.jo

1 Faculty of Engineering and Natural Sciences, Malatya Turgut Ozal University, 44210 Malatya, Turkey

2 Faculty of Computer Sciences and Informatics, Amman Arab University, Amman 11953, Jordan

by employing machine learning algorithms, which are executed before the testing stage to reveal which modules are being tested in detail [5].

Recently, the application of deep neural networks to discover vulnerable software components is performed based on using gated recurrent unit (GRU), long short-term memory (LSTM), and recurrent neural networks (RNNs) to learn sequences over time intervals [6]. These networks can process memories of sequential features due to storing previous inputs in the networks' internal state and mapping the complete history of previous inputs to target vectors in principal. In this paper [7], the first deep learning-based vulnerability detection system, aiming to help human experts with the monotonous and subjective work of manual feature defining and decreasing the false negatives incurred by other vulnerability detection systems developed. According to the experiments, VulDeePecker was capable of achieving a significantly lower false-negative rate in comparison with other vulnerability detection systems, in addition to helping human experts with defining features manually [8].

In this paper [9], a genetic feature selection model is employed to reveal optimal feature sets to enhance static analysis alerts' classification accuracy. The proposed method is based on three open-source PHP applications. In [10], the DNN model is utilized to construct an automatic vulnerability classifier to achieve efficient vulnerability classification. The National Vulnerability Database of the United States is being used to validate the efficiency of the suggested model. In this paper [11], a hybrid approach is proposed, which combined a genetic algorithm (GA) for feature optimization and a DNN for classification. Furthermore, the DNN technique is improvised by utilizing an adaptive auto-encoder that better represents the chosen software features. The experimental analysis showed that the proposed method obtained enhanced KC1, CM1, PC3, and PC4 datasets. In [12], deep learning techniques used for anomaly based network intrusion detection are examined—comparing the detected anomalies with the traditional machine learning techniques, e.g., random forest, SVM, and AdaBoosting was made. Other optimizers can be used in vulnerability detection such as arithmetic optimization algorithm [13] and Aquile optimizer [14].

In this paper [15], an artificial neural network-based methodology is suggested for anomaly detection adjusted to the Apache Spark in-memory processing platform. The proposed method is assessed compared to three standard machine learning algorithms, including nearest neighbor, decision trees, and support vector machines. The findings confirmed that the proposed method performed better than other forms, usually obtaining F-scores of 98–99% and providing significantly higher accuracy than alternative techniques. In [16], a review of the current denial-of-service (DoS) attack and defense concepts is performed from theoretical and practical aspects. This paper focuses on ensuring both a fresh and relevant state-of-art reference with the various perspectives, e.g., economic denial-of-service (EDoS) attack or offensive countermeasures in cyberspace. In [17], the SU-genetic method, targeting massive network traffic of DDoS attacks, is suggested to select significant features of original attack data. The proposed SU-genetic feature selection method enhanced all three classification-based detections (i.e., BayesNet, J48, and RandomTree).

The performance of vulnerability prediction models is significantly affected by static characteristics utilized as predictors and how the said features are used in statistical learning machinery aspects. Static analysis (SA) tools represent the static scanning of code or binaries to locate bugs, defects, flaws, or security vulnerabilities [18]. Currently, SA tools provide developers with many false positive and unactionable alerts. Unfortunately, the alerts generated by those tools are not always correct, and a high number of false positives is one of the primary reasons for not using these tools widely. There is no consensus on what features are relevant in the SA domain, and we should focus on how to select the relevant features. In the literature, most of the published works utilized various feature selection methods include deep learning, conventional feature selection, and the classification of vulnerable software components. This research aims to develop and assess the feature selection methods that help improve the classification accuracy of a vulnerability prediction model [19–21].

In this paper, an improved feature selection method is proposed to reveal optimal feature sets by enhancing the classification accuracy of static analysis. This paper's proposed method is the immune-based feature selection that is utilized to discover optimal feature sets by improving the classification accuracy of static analysis. An immune feature selection method is proposed as a potential solution in the fixed analysis domain. Experiments are conducted on the three standard datasets. The detailed assessment of the impacts of the proposed artificial immune system dynamics on the prediction of software vulnerabilities, the design of a novel feature selection algorithm for an SVP problem, and the assessment of seven classification algorithms on three public datasets constitute the contributions of this article. The proposed method improved the current dataset generation framework to involve additional predictive software features. So, a real-world static analysis dataset is used based on three open-source PHP applications. The proposed feature selection method is presented and tested to determine the benefit rate of choosing relevant features and to study the influence on the classification of vulnerable components. An in-depth investigation of the proposed feature selection method to enhance vulnerability classification is provided. Compared to conventional immune-based algorithms, finding a better feature subset for classification with immune algorithm-based feature selection is possible. The obtained results showed that the proposed feature selection method achieved better results compared to other comparative methods. Moreover, the proposed method improved the usefulness of static analysis tools by learning optimized historical data features.

The principal contributions of this paper are given as follows:

- This paper is the first model that defines features by using the clustering theory of the clonal selection algorithm for software vulnerability prediction problems. This makes this methodology a novel approach to effective detection and analysis of security vulnerabilities.
- This paper introduces a new feature selection method based on deep learning models. The selected features are utilized in predicting software vulnerability.
- The immune-based feature selection method is utilized to discover optimal feature sets by increasing the classification accuracy and decrease the false- positive rate of static analysis.
- We exploited long short-term memory (LSTM) and gated recurrent unit (GRU) to capture the long context correlations in terms of deep-learned long-lived team-hacker features.
- We investigate how to utilize a deep neural network (DNN) recurrent neural network models to predict vulnerabilities to learn software metrics.

The rest of this paper is organized as follows. Section 2 explains the background of the employed models. The problem definition and procedure are described in Sect. 3. The proposed method is presented in Sect. 4. The tests and performance findings are shown in Sect. 5. The conclusion and future studies are presented in Sect. 6.

2 Background

In this section, the background of the most common techniques that have been used in the literature is given as follows.

2.1 Static analysis (SA) tool

Static analysis is a powerful method to find issues in software without its execution [22]. Static analysis, which is also called code analysis, is generally performed in the code review course. The static analysis represents a process of system or component examination, considering its form, content, structure, or documentation without executing the code. The transformation of a source code into an intermediate representation or model constitutes a very frequently employed static analysis approach. Control flow graph (CFG), abstract syntax tree (AST), and program dependency graph (PDG) are among classic examples. This is especially important since it can show potential problems early, making them cheaper and easier to fix. Static analysis (SA) tools are simple to style checkers or follow intricate rules to find developers' issues often overlook efficiently. In other words, it searches for problems in the application based on a predetermined set of rules, representing possible anomalies that usually occur in the code. The collection of rules comprises various errors ranging from mistakes in the source code to complex errors in the system's logic. They analyze source code to find flaws and defects without the need to execute the binaries.

2.2 Biological immune system

The immune system represents a complicated network of organs and cells involved in defense of an organism against pathogens. An essential mechanism of the immune system recognizes its own (self) and foreign (nonself) cells. In this way, it can establish a defense against the attacker instead of self-destruction. The innate or non-specific immune system, present in an organism from the time of its birth and not adapting throughout its life, represents the first barrier of defense against pathogens. Unlike the innate system, the adaptive immune system, which means the second barrier, can recognize and remember particular pathogens. B-cells and T-cells are the two major classes of lymphocytes. The formation of amazingly diversified B-cells and T-cells constitutes the human immune response [23]. A unique genetic mechanism forms the specificity of B-cell receptors and T-cell receptors. T-lymphocytes, the subsets of memory cells, ensure their functions, e.g., central memory, regulatory memory, effector memory, tissue-resident memory, and stem memory T-cells. Biologically, the primary immune response occurs if the immune system meets with a foreign substance for the first time, and the importance removed from the procedure takes place. Thus, a particular number of B-cells remain in the immune system and function as immunological memory. The reason for this is to ensure that the immune system launches a quicker and more powerful attack against the infecting agent, named the secondary immune response.

2.3 Importance of artificial immune systems for static analysis

Artificial immune systems (AISs) represent problem-solvers inspired by biology and utilized successfully as vulnerability detection systems [24]. Following an immune response to a particular antigen, some sets of cells and molecules are endowed with increased life spans to ensure quicker and more robust immune responses to infections in the future by identical or similar antigens. Immunological memory constitutes an exclusive characteristic of the immune system since it is capable of "storing" data on a stimulus and mounting an efficient response in case of reencountering the inspiration. Static analysis tools have the potential to create alerts to the quality and security vulnerabilities of an application. Nevertheless, they introduce a lot of false positives and unactionable alerts to analysts and developers. An alert's lifetime is an exciting property since it can be assumed that a short lifetime means that the alert is found and important enough to fix it quickly. Deep neural network algorithms are used to discover deep-learned feature sets relevant to the accurate classification of static analysis alerts [25, 26]. Thus, they aim to detect historical features and realistic source code metrics.

2.4 Feature selection

Feature selection is the process of determining which features are relevant to or will improve classification accuracy [27]. Furthermore, feature selection includes identifying a subset of the most beneficial features generating consistent outcomes as the whole original set of features. In the problem of feature selection, a learning algorithm encounters the problem of choosing an optimized subset of original features. It has been demonstrated that many problems related to feature selection are NP-hard. Three approaches have been employed to deal with feature selection: filter, wrapper, embedded, and, more recently, hybrid methods. Filter methods do not consider learning algorithms but use statistical measures to determine feature importance. Features are ranked and either kept or removed from the dataset. Wrapper methods take the selected feature sets and evaluate their accuracy on a predictive model. Each subset is utilized for training a model and then testing it. Classification accuracy is used to score the subset's performance. Embedded methods perform feature selection and classification simultaneously. Hybrid models are a combination of filter, wrapper, and/or embedded methods [28].

3 Problem definition and procedure

3.1 AIS-based classifiers

The immune system has high distribution; it is adaptive and self-organizing by its nature, preserves previous meetings' memory, and continuously learns novel arrangements [23]. The immune system is capable of recognizing, identifying, and responding to various patterns.

3.1.1 Clonal algorithm (CLONALG)

The clonal selection principle represents the term used to describe the primary features of an adaptive immune response to an antigenic stimulus [23]. The central concept of clonal selection theory is how a B-cell responds to an invaded antigen by altering a receptor known as an antibody. The general one, CLOGNALG, is one of the clonal selection algorithms' members. The clonal selection algorithm flow stages are selection, cloning, mutation, reelection, and replacement. Clonal selection algorithm flow mainly includes steps of selection, cloning, mutation, reelection, and replacement. The algorithm's goal is to develop a memory pool of antibodies that represents a solution to an engineering problem. It presents the opinion that the exclusive proliferation of cells capable of recognizing an antigenic stimulus will occur, and their selection will take place. The algorithm provides two mechanisms for searching for the desired final pool of memory antibodies. The first is a local search provided via affinity maturation (hypermutation) of cloned antibodies. More clones are produced for better-matched (selected) antibodies. This allows those antibodies with low specificity with the antigen a broader area in the domain which to maturate. The second search mechanism provides a global scope. It involves the insertion of randomly generated antibodies to be inserted into the population to increase diversity further and provide a means for potentially escaping local optima. This paper found a new scheme to replace the current clonal selection algorithm to improve the algorithm's overall search accuracy and efficiency to improve the update efficiency of the antibody collection.

The clonal selection theory of immunity, as shown in Fig. 1, has been identified [29]. It explains the cloning and activation of only specific lymphocytes that can fight the particular infection after the pathogen. The CLONALG algorithm bears similarities to the algorithm of evolutionary strategies from the area of evolutionary computation. Although CLONALG performs a search by blind cumulative variation and selection by cloning, mutating, and advancing the best match recognition cells. Clonal selection functions on T-cells as well as B-cells. Learning

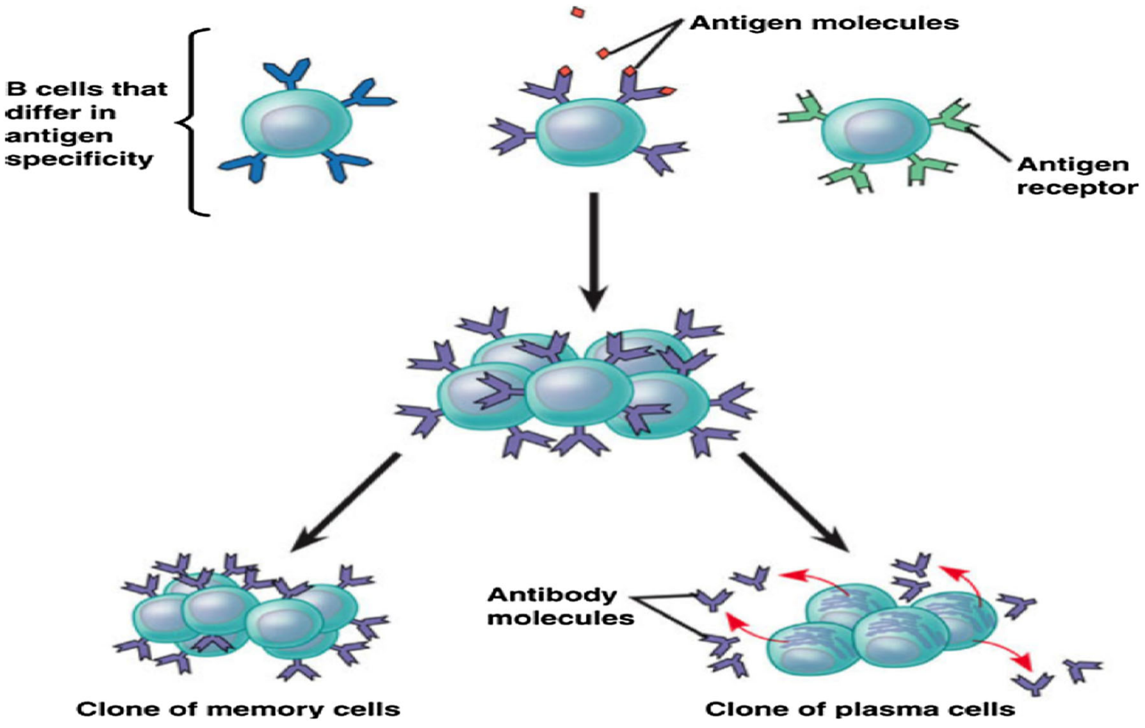


Fig. 1 The clonal selection theory of ımmunity [29]

through clonal selection includes increasing the population size and the cells' affinity in question that have confirmed themselves to have a value at the antigen recognition stage.

3.1.2 Clonal selection classifier algorithm (CSCA)

The clonal selection classification algorithm (CSCA) is a novel classification algorithm based on abstractions of the clonal selection theory of acquired immunity and the CLONALG technique and its variations. The proposed algorithm attempts to design and invent an efficient, convenient classification algorithm inspired by clonal selection. Implementing the CSCA is presented as a plug-in for the WEKA machine learning workbench, like a naïve application of the CLONALG algorithm for classification. The clonal selection classifier algorithm (CSCA) aims to maximize classification accuracy and minimize misclassification accuracy [19]. The principle of clonal selection also forms the basis of the CSCA.

3.1.3 Immunos algorithms

The Immunos-1 algorithm represents an artificial immune system-based algorithm. Thus, the obtained clone population is kept for the classification of novel data instances. It assumes a single problem domain, which consequently turns the T-cells and amino acid library's requirement into an absolute one. For every classification label, a B-cell clone population is formed, while every antigen in the training set is included. At the classification stage, an avidity value is found for every B-cell clone population. According to the avidity value, the assignment of a data instance is performed to a class of clones with the maximum avidity value, which is found by computing the affinity between the specific antigen and every single cell of clone shown in Eq. (1).

$$\text{affinity} = \sqrt{\sum_{A}^{i=1} af_i} \tag{1}$$

where A refers to the total number of features in the data vector, and af_i denotes the affinity value of the ith feature. The computation of the affinity measure between numeric features is performed using the Euclidean distance. The calculation of the avidity measure is performed using Eq. (2).

$$\text{avidity} = \frac{\text{CS}}{\text{CA}} \tag{2}$$

where CS refers to the total number of B-cells in the clone population, while CA refers to the clone affinity.

Immunos-99 represents a classification algorithm based on the principles of artificial immune systems (AISs). It is inspired by the Immunos family, which includes additional immune-inspired methods, including cell proliferation and hypermutation. The training stage of the Immunos-99 algorithm begins with dividing data into antigen groups based on classification labels. Then, a B-cell population is obtained from every antigen group [30]. The formation of people is performed in an independent way from other obtained populations. Afterward, the B-cell population is subjected to all antigens for all groups. During the mentioned process, the affinity measure between every B-cell and the antigen is found. The population's ordering is carried out based on the declared affinity values, followed by computing fitness scores. The fitness score indicates the usefulness of a particular B-cell.

3.1.4 Artificial immune recognition systems (AIRS)

According to the computed fitness values, the application of pruning to the population is performed to remain exclusively healthy. After pruning on the population, affinity maturation is carried out to improve the population specificity to its particular antigen group [31]. The artificial immune recognition system (AIRS) is among this kind of supervised learning AIS that has succeeded considerably in a wide variety of classification problems.

The versions of AIRS algorithms are AIRS1, AIRS2, and AIRS parallel algorithms. The primary stages of the AIRS1 algorithm include the first version of AIRS, and the AIRS2 algorithm represents the second version. Whereas AIRS1 utilizes the mutation parameter that the user can define, AIRS2 utilizes the somatic hypermutation concept, in which the clone's mutation ratio is proportional to the affinity. While the clone's classes can alter after mutation in the AIRS1 algorithm, types are not permitted to change in the AIRS2 algorithm. The parallel AIRS version shows the distributed character of immune systems and their similar processing features [32]. In the beginning, the assignment of every part of the training dataset is performed to the number of processes. Therefore, it is provided that the number of the memory pool is formed as a result of the execution of the AIRS algorithm on every cycle. Accordingly, the merging of the acquired memory pools is ensured.

3.2 Deep learning-based classifiers

Deep learning (DL) represents a branch of machine learning models. It is characterized by its ability to extract hierarchical representations from input data due to the establishment of deep neural networks with multiple layers of nonlinear transformations.

3.2.1 Recurrent neural network (RNN)

Recurrent neural networks (RNNs) can memorize arbitrary length sequences of input patterns by establishing relations between units [33]. The transition function at every time step (t) takes the current time information, which is denoted as the previously hidden output. The updating of the recent hidden work is performed using Eq. (3).

$$h_t = \mathbb{H}(X_t + h_{t-1}) \tag{3}$$

where $\mathbb{H}$ refers to a nonlinear and differentiable transformation function. When the complete sequence is processed, the hidden output at the final time step, i.e., ht, may be considered a sequential data vector. The supervised learning layer's addition is performed to map the acquired representation ht to targets. It is possible to train the model via backpropagation through time.

3.2.2 Gated recurrent unit (GRU)

GRU is an improved version of standard recurrent neural networks (RNNs) [31]. GRU networks have two gates: a reset gate (r), which performs the adjustment of the incorporation of novel input with the previous memory, and an update gate (z), which serves as the control of preserving the last memory. The transition functions in the hidden units of GRUs are presented in Fig. 2.

The update gate helps the model determine the amount of the previous information (from past time steps). Equation (4) is used for the current update gate.

$$Z_t^j = \sigma(W_z X_t + U_z h_{t-1})^j \tag{4}$$

If X_t is plugged into the network unit, it is multiplied by its weight $W(z)$. Identically, it is applicable to h_{t-1} that holds the information for the past $t-1$ units, and it is multiplied by its weight $U(z)$. A sigmoid activation function is applied to squash the outcome between 0 and 1. The function behavior is presented in Fig. 3.

The reset gate is utilized for the model to make a decision on what amount of the past information it is necessary to forget. Equation (5) is used to update the rest gates.

$$r_t^j = \sigma(W_r X_t + U_r h_{t-1})^j \tag{5}$$

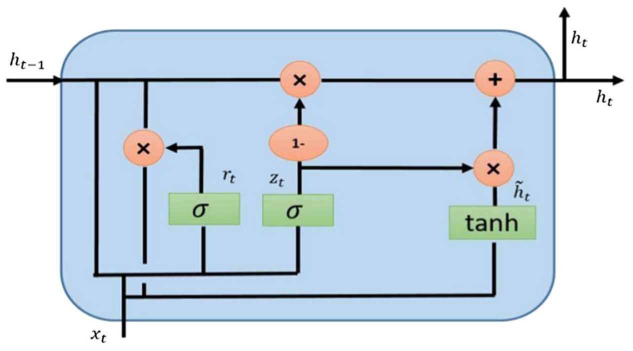


Fig. 2 GRU architecture [31]

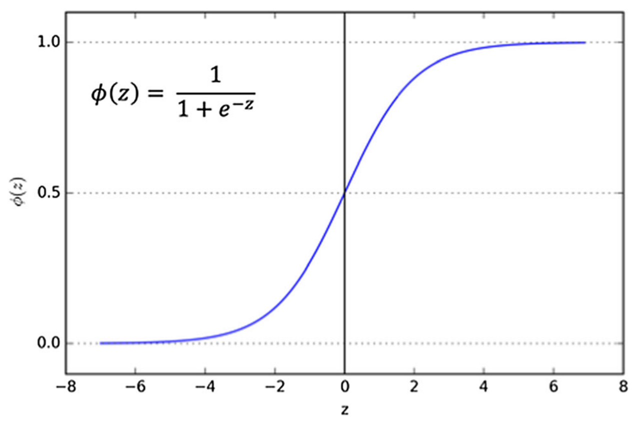


Fig. 3 The convergence of sigmoid activation function

The mentioned equation is similar to the update gate's equation. The difference is due to the weights and the usage of the gate. This will be observed in the following sections to show how the gates' final output will be affected precisely. At first, let us start with the reset gate's usage. t represents a new memory content, making use of the reset gate to store the relevant information from the last time. It is computed in the following equation:

$$\hat{h}_t^j = \tanh(WX_t + r_t \odot Uh_{t-1}) \tag{6}$$

Finally, there is a need for the network to calculate the h_t vector that keeps information for the current unit. The update gate determines what must be collected from the current memory content $\hat{h}_t$ and what from the past steps h_{t-1}, final memory at the current time step.

$$h_t^j = \tanh\left(z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t\right) \tag{7}$$

The architectures of the GRU and LSTM memory block are presented in Figs. 3 and 4, respectively.

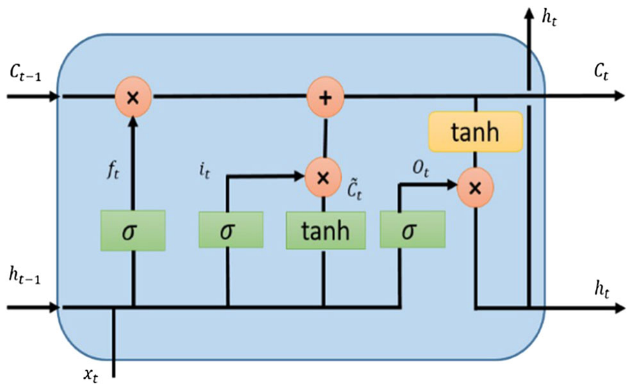


Fig. 4 LSTM architecture [31]

3.2.3 Long short-term memory (LSTM)

LSTM networks are one of the most effective solutions to a sequence of prediction problems because of the recognizing patterns in data sequences [34]. Since LSTM networks have a particular type of memory, they can selectively remember ways for a long time. They represent quite a reasonable approach to predicting the period with the unknown long delays between essential events. The LSTM memory block's structure is composed of three gates and a self-recurrent connection. The gates are an input gate, forget gate, and output gate responsible for remembering things and manipulating the memory cell. These gates perform the modulation of interactions between the memory cell itself and its environment. The input gate adds information to the cell state. The forget gate ensures that the cell remembers or forgets the previous state of the memory cell. The output gate selects valuable data from the current state of the cell and shows it out.

$C^j{}_t$ refers to the memory amount of every *j*th LSTM unit at *t* time. The output of the *j*th LSTM at *t* time is represented by the $h^j{}_t$ LSTM unit, which is calculated by Eq. (8).

$$h_t^j = \sigma_t^j \tanh\left(c_t^j\right) \tag{8}$$

$\sigma^j{}_t$ denotes the output gate managing the memory content exposure. The output gate is presented as follows:

$$\sigma_t^j = \sigma(W_0X_t + U_0h_{t-1} + V_0C_t)^j \tag{9}$$

σ denotes the sigmoid function, and V_0 represents a diagonal matrix. $\hat{C}_j{}^t$ refers to a new memory content of the memory unit, updated by partially forgetting the current memory and adding the novel memory content to $C_j{}^t$.

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \hat{C}_t^j \tag{10}$$

The new memory contents are presented below:

$$\hat{C}_t^j = \tanh(W_cX_t + U_ch_{t-1})^j \tag{11}$$

$f^j{}_t$ modulates the current memory forgetting gate. An input port $i^j{}_t$ modulates the addition degree of the novel memory content to the memory cell. V_f and V_i represent diagonal matrices.

$$f_t^j = \sigma\left(W_fX_t + U_fh_{t-1} + V_fC_{t-1}\right)^j \tag{12}$$

$$i_t^j = \sigma(W_iX_t + ih_{t-1} + V_iC_{t-1})^j \tag{13}$$

4 The proposed method

Bio-inspired computing represents an active area because its nature is solving many real-world problems. It uses vulnerability prediction modeling (VPM) techniques and leverage predictive modeling techniques as vulnerable code components. The mentioned techniques depend on source code features as indicators of vulnerabilities for the establishment of prediction models. VPM uses supervised learning techniques to build prediction models based on a set of features and then uses them to evaluate the likelihood of software components being vulnerable. To discuss the problem of high false-positive rates, the current study aimed to develop and evaluate methods for feature selection that help to enhance the classification accuracy of static analysis.

Natural computation algorithms are commonly applied in feature selection for performance enhancement and reduction of the feature dimension. Feature selection is generally employed to achieve identical or higher performance by utilizing a lower number of features. It may be regarded as an optimization problem and aims to determine an optimal feature subset from the existing features by a particular criterion function. The clonal selection algorithm is an excellent option to solve an optimization problem. It presents the mechanisms of affinity maturation, clone, and memorization.

Different strategies are recommended to understand the learning and memory of the immune system. Based on the hypothesis in [35], in the case of the activation of the artificial recognition ball (ARB), such as B or T-cells, by an antigen, there is a period when they are expected to meet the identical antigen. ARB cells remain in memory for weeks or even months. Lymphocytes are capable of maintaining the dormant state for long years in a similar time interval. The antigen itself or its shape that is partially deformed (a small alteration) accumulates in organs or lymph nodes. The immune system's memory, exposed to the antigen, is fed at fixed intervals.

Nevertheless, survivors are not known if the antigen has not activated ARB cells for long years. A basic theory that supports the mentioned opinion is based on the immunological theory. By the idea in question, some immunologists argue that a particular internal restimulation mechanism ensures the protection of the immune memory for a prolonged time. An exciting study utilized an immune-based feature selection methodology by suggesting long-lived team hackers for pattern recognition and optimization purposes used for classification. The clonal selection theory explains the process of memory. This theory says that essentially the antigen "selects" the correct B-cell to produce antibodies that will be used against it and will destroy infected cells. The selected B- or T-cells will then be cloned, but some are not activated and instead adopt a long-term memory form.

In this paper, the effect of the immune system on software vulnerability is studied and analyzed. The analysis of software vulnerabilities is performed using long-lived team hackers known as an intelligent antibody to find vulnerabilities based on the artificial immune system (AIS), as shown in Fig. 5. A new framework, named the long-lived team- hackers-based deep neural network (LLTH-DNN), was built as the software's generalized vulnerable components. The LLTH-DNN is evaluated as a lifetime-based mechanism to discover historical data for classification as vulnerable or not vulnerable. Figure 6 shows the flowchart of the LLTH-DNN model. Algorithm 1 shows the pseudocode of the proposed LLTH-DNN algorithm.

4.1 Representation

Each data were assumed to be denoted as $x = [\times\ 1, \times\ 2, \ldots, xn]$, where n is the length of the data sample, and training data have a corresponding target value such as vulnerable or not vulnerable, defined in software vulnerability models. Binary encoding is used to represent feature selection or exclusion in the solution set. Each antigenic pattern, a candidate solution, is represented as a bit string with length n, where n is the total number of features. The jth feature was retained if the jth bit was 1 and removed if the jth bit was 0.

4.2 Fitness function

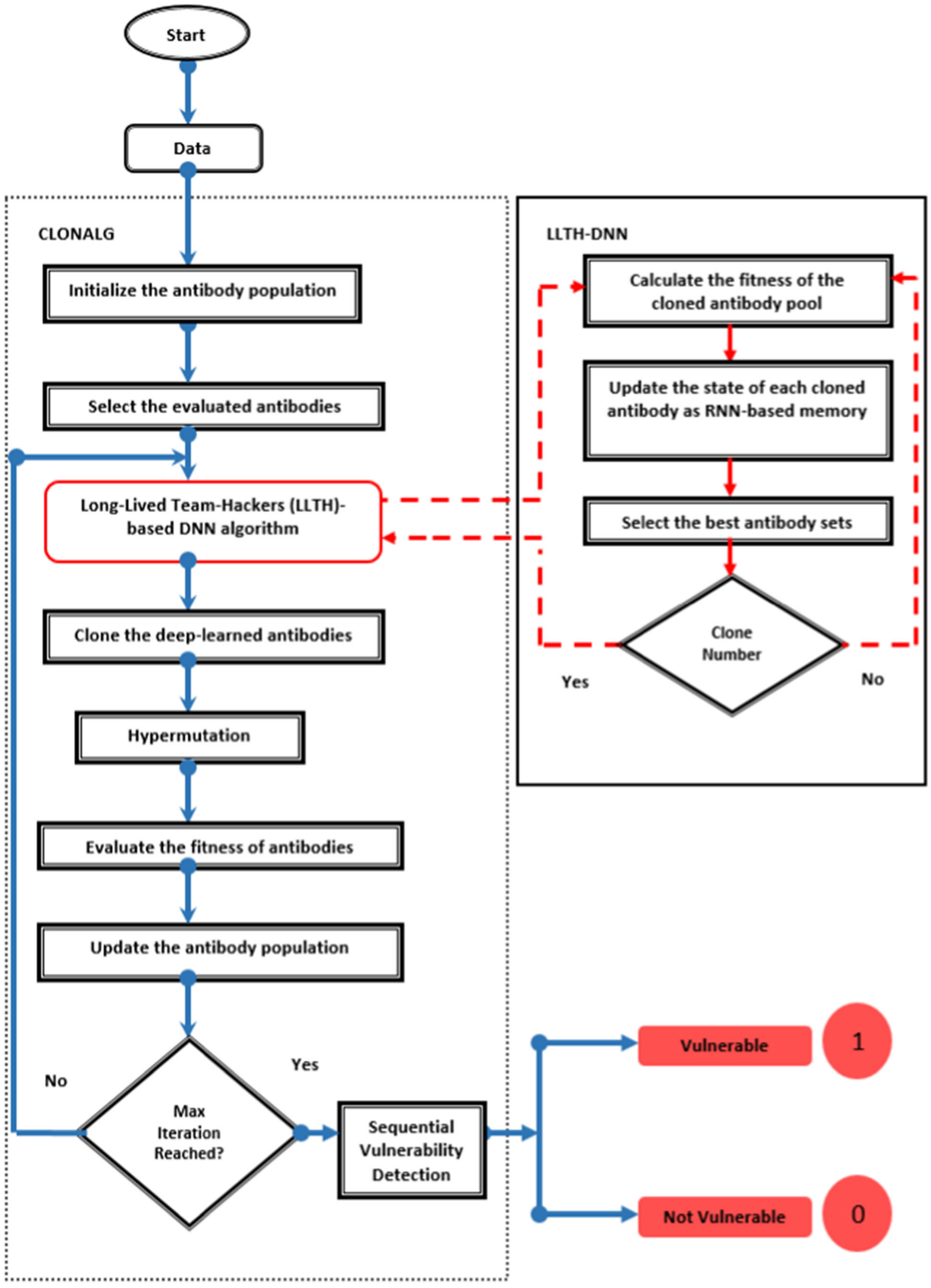
An antigenic pattern's fitness is proportional to the model's classification accuracy on the test set using the selected subset of features. The fitness function is used to reveal the quality of every antigenic pattern.

DNN methods take a sequence $\{\times\ 1, \times\ 2, \ldots, xT\}$ as input and construct a corresponding sequence of hidden states (or representations) $\{h1, h2, \ldots, hT\}$. The antigenic pattern with the maximum values of fitness for vulnerabilities may be regarded as the most representative example of a population of vulnerabilities. The algorithms' fitness function was calculated according to the accuracy of the random forest classifier equation.

4.3 Long-lived team-hackers selection method-based deep neural network (LLTH-DNN)

Feature selection methodologies were developed and evaluated for the improved classification accuracy of static analysis. After data were prepared and preprocessed, the first step in the model was to perform feature selection. For the feature selection component, immune-based algorithms were employed.

Long-lived team-hackers selection consists of two repertoires, such as an antibody set and an antigen set. Ab's repertoire is produced randomly and includes two subsets Abm (the memory repertoire with size m) and Abr (the remaining repertoire with size r). The memory repertoire represents the output of the algorithm shown below. The


Start
Data
CLONALG
Initialize the antibody population
Select the evaluated antibodies
Long-Lived Team-Hackers (LLTH)-based DNN algorithm
Clone the deep-learned antibodies
Hypermutation
Evaluate the fitness of antibodies
Update the antibody population
No
Yes
Max Iteration Reached?
Sequential Vulnerability Detection
LLTH-DNN
Calculate the fitness of the cloned antibody pool
Update the state of each cloned antibody as RNN-based memory
Select the best antibody sets
Clone Number
Yes
No
Vulnerable
1
Not Vulnerable
0

◀ **Fig. 5** Suggested flowchart of the LLTH-DNN model

antigenic pattern is valuable for analyzing predefined vulnerability properties. Using deep learning methods (GRU and LSTM) allows the relative detection of vulnerable metrics. Due to the immune system's autonomy and immutability, it is possible to detect features relatively by DNN as a series of vulnerabilities that occur during its lifetime.

Algorithm: Pseudocode of the LLTH-DNN Algorithm

```
1- {InitializeAntibodyPool (AntibodySet)}
2- {InitializeFeatureSet($\Omega$)}
3- [Train] {1…N} (Input Size)
4- {IntroduceAntibodyPool} ($Ab_N$)
5- FOR I ← Iteration Number DO
6- Affinities ← {calcAffinities ($Ab_i$)}
7- Clonenum ← Select{Affinity ($Ab_i$)}
8- Fitness ← Accuracy (Feature set ($\Omega$), Abi)
9- AntibodyDNN ← CreateDNNMemory (Abi, t,State_id)
10- FOR (Abi $\in$ Clonenum)
11- ClonesAntibodies ← (CloneandHypermutated (Abi))
12- CloneAffinities ← {calcAffinities (Abi)}
13- Fitness ← Accuracy (Feature set ($\Omega$), Abi)
14- {UpdateAntibodyPool} ← (CloneAffinities)
15- CDNN ← UpdateDNNMemory (CDNN, t,State_id)
16- END FOR
17- bestAffinity ← {getBestAffinity (CloneAffinities)}
18- $\Omega^*$ ← NewFeatureSet(CDNN, bestAffinity)
19- END FOR
```

In the proposed framework, an enhanced clonal selection algorithm based on DNN methods is proposed. The repertoire CDNN undergoes the process of affinity maturation. Affinity maturation represents a simple mutation of the clone population CDNN in proportion to its affinity.

5 Results and analysis

This section describes the experimental performances of the algorithms for vulnerability detection. To show the proposed algorithms' efficiency, versus AIS-based classifiers and six machine learning classifiers are applied. Also, three different experiments were conducted on public phpMyAdmin, Moodle, and Drupal datasets.

A public dataset containing data on security vulnerabilities is used. Machine learning features of three open-source PHP applications referred to as the PHP security vulnerability dataset are employed. The complete raw dataset, replication dataset, and all scripts used to create datasets can be downloaded from https://seam.cs.umd.edu/webvuldata. The dataset contains 233 verified security vulnerabilities. The data obtained from 95 versions of phpMyAdmin 3.3.0, Moodle 2.2.0 version, and the version of Drupal 6.0.0. source code metrics were collected and included in the dataset. Metrics were included and linked to file names for each project and performance. These include the following: lines of code, non-HTML code, cyclomatic complexity, number of functions, Halstead's volume, maximum nesting complexity, fan-in, fan-out, total external calls, internal processes, external procedures, and external calls to function.

A feature selection method is developed and evaluated to investigate the improved classification accuracy of static analysis. The technique used for the feature selection model is an embedded model. The model's selected feature subsets and classification performance metrics are exported for each experiment. In the experimental process, tenfold cross-validation is employed for assessment. Using the scheme mentioned above, the original dataset is separated randomly into ten subsamples of equal size. For every time, a single subsample is utilized for validation purposes, while the remaining nine subsamples are kept for training purposes. The process is performed ten times, and the average is taken. The AIS was run with the following setting parameters: affinity threshold value, clonal ratio, mutation ratio, np, comprehensive resource, stimulating value, hypermutation ratio, and the number of tests, and max iteration number as 0.2, 10, 0.1, 2, 150, 0.9, 2.0, 10, and 100, respectively, and the random forest was used as the classifier. To assess classification algorithms' performance, statistical evaluation metrics, precision, recall, F-measure, classification accuracy, true positive rate, and false-positive rate measure values are employed. Weka is used to obtain classification accuracies. The Wekaclassalgos project source code is used.

5.1 Evaluation measures

Classification accuracy represents the proportion of true positives and true negatives is calculated by Eq. (14) [36, 37].

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \quad (14)$$

where TN denotes the number of true negatives, TP refers to the number of true positives, FP represents false positives, and FN represents false negatives. The precision equation is given in Eq. (15).

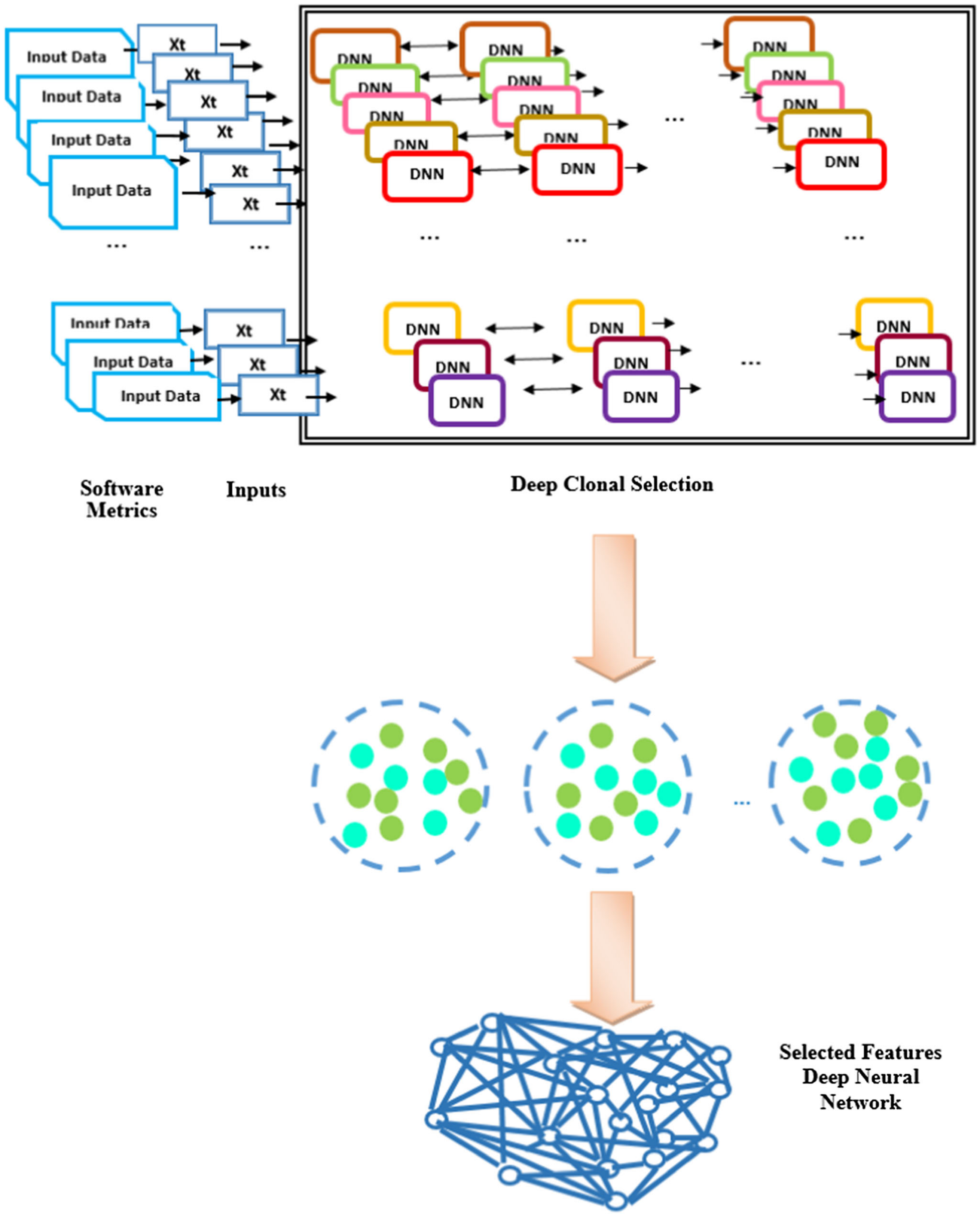


Fig. 6 Proposed LLTH-DNN model

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (15)$$

The recall represents the ratio of true positives to true positives and false positives, as shown in Eq. (16) [38].

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (16)$$

The value of F-measure ranges from 0 to 1. It represents a harmonic mean of precision and recall and is given in Eq. (17) [39].

$$\text{F - measure} = \frac{2{*}\text{Precision}{*}\text{Recall}}{\text{Precision} + \text{Recall}} \quad (17)$$

5.2 Results and discussion

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk symbol after the value in all tables. We focus on comparing the statistical performance of the algorithms in each experiment.

In experiment 1, the statistical performance of the algorithms was tested on the phpMyAdmin version 3.3.0 dataset. According to the results in Table 1, comparing the classification accuracy on the phpMyAdmin dataset shows that the LLTH-GRU algorithm can significantly improve classifiers' performance and reduce the false-positive rates. As shown in this table, according to AIS-based classifiers, the classification accuracy increases by about 8.12% for the LLTH-GRU algorithm and by 5.89% for the LLTH-LSTM algorithm. In terms of the true positive rate, the LLTH-GRU algorithm's performance on the phpMyAdmin dataset was 0.960. The best precision performance was achieved, especially by the LLTH-GRU algorithm. Concerning the false-positive rates of the LLTH-GRU and LLTH-LSTM algorithms, they were 0.652 and 0.798, respectively. However, the LLTH-LSTM algorithm had an accuracy of 93.77% and a true positive rate of 0.938. The results indicated that both the LLTH-GRU and LLTH-LSTM algorithms had a high true positive rate. The experimental results demonstrated that the Immunos-1 algorithm obtained the worst accuracy performance of 24.2%. Furthermore, the PAIRS algorithm achieved the best classification performance with 87.88% accuracy. Therefore, we can conclude that the two suggested algorithms are more effective in detecting security vulnerability.

Comparing the results obtained from the six classifiers in Table 2 shows that the XGBoost outperforms the other seven classifiers. The MLP and AdaBoost classifiers are better than the SVM and J48. Furthermore, the NB classifier exhibited the worst classification performance in experiment 1. Comparing the classification performances in these cases demonstrates that the suggested deep learning-based feature selection algorithms can significantly improve classifiers' performance and reduce the false-positive rates.

The statistical performance of the algorithms on the Moodle version 2.2.0 dataset was tested in experiment 2. According to the performance results in Table 3, the LLTH-GRU and LLTH-LSTM algorithms almost outperformed the other algorithms (CSCA, AIRS1, AIRS2, and PAIRS). They achieved 99.15% and 98.78% classification accuracies with 0.992 and 0.988 true positive rates. However, compared to the false-positive rates, the suggested algorithms achieved 0.992 and 0.981 comparative results, respectively. This table shows that the classification accuracy increased by about 10.25% for the LLTH-GRU algorithm and by 9.88% for the LLTH-LSTM algorithm compared to AIS-based classifiers. The standard AIS-based classifiers exhibited comparable classification accuracies, except for Immunos-1 and Immunos-99, in experiment 2. According to the results, the worst accuracy performance of 32.6% was obtained by the Immunos-1 algorithm.

In Table 4, we compared the performance of standard classifiers in vulnerability detection. The obtained results

Table 1 Performance of AIS-based classifiers versus suggested algorithms in experiment 1

Statistical metrics	AIS-based classifiers							Suggested algorithms	
	CSCA	Immunos-1	Immunos-2	Immunos-99	AIRS1	AIRS2	PAIRS	LLTH-GRU	LLTH-LSTM
P	0.825	0.84	0.839	0.844	0.871	0.857	0.876	0.962	0.918
R	0.831	0.242	0.916	0.389	0.858	0.783	0.879	0.960	0.938
F	0.813	0.302	0.876	0.489	0.865	0.816	0.876	0.950	0.935
Accuracy	83.1	*24.2**	81.61	38.9	85.83	78.2	87.88	**96**	93.77
TPR	0.831	0.242	0.816	0.389	0.858	0.783	0.879	0.960	0.938
FPR	0.699	0.251	0.816	0.399	0.686	0.693	0.704	0.652	0.798
AUC	0.843	0.335	0.832	0.453	0.882	0.804	0.894	0.975	0.954

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk and italics

Table 2 Performance of standard classifiers in experiment 1

Statistical metrics	Standard classifiers						XGBoost	AdaBoost
	MLP	RF	k-NN	SVM	NB	J48		
P	0.905	0.922	0.893	NaN	0.878	0.886	0.912	0.905
R	0.992	0.929	0.891	0.916	0.854	0.910	0.904	0.903
F	0.899	0.907	0.892	NaN	0.865	0.894	0.916	0.917
Accuracy	92.23	92.85	89.13	91.61	*85.4**	90.9	**93.5**	92.1
TPR	0.922	0.929	0.891	0.916	0.854	0.748	0.935	0.921
FPR	0.781	0.747	0.582	0.916	0.619	0.291	0.657	0.712
AUC	0.945	0.957	0.913	0.932	0.874	0.934	0.927	0.915

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk and italics

Table 3 Performance of AIS-based classifiers versus suggested algorithms in experiment 2

Statistical metrics	AIS-based classifiers							Suggested algorithms	
	CSCA	Immunos-1	Immunos-2	Immunos-99	AIRS1	AIRS2	PAIRS	LLTH-GRU	LLTH-LSTM
P	0.884	0.882	0.884	0.992	0.884	0.884	0.884	0.984	0.984
R	0.882	0.327	0.882	0.36	0.854	0.887	0.889	0.992	0.988
F	0.888	0.482	0.888	0.52	0.884	0.886	0.887	0.988	0.986
Accuracy	88.1	*32.6**	88.1	36.0	88.4	88.6	88.9	**99.15**	98.78
TPR	0.882	0.327	0.882	0.36	0.884	88.6	0.889	0.992	0.988
FPR	0.882	0.006	0.882	0.022	0.844	86.7	0.884	0.992	0.981
AUC	0.892	0.354	0.896	0.385	0.896	0.894	0.903	0.996	0.985

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk and italics

Table 4 Performance of standard classifiers in experiment 2

Statistical metrics	Standard classifiers						XGBoost	AdaBoost
	MLP	RF	k-NN	SVM	NB	J48		
P	0.984	0.984	0.984	NAN	0.986	NAN	0.973	0.953
R	0.991	0.992	0.982	0.992	0.993	0.992	0.946	0.968
F	0.987	0.998	0.988	NAN	0.958	NAN	0.932	0.967
Accuracy	99.11	99.15	98.23	**99.18**	*93.33**	99.1	99.16	98.6
TPR	0.991	0.992	0.982	0.992	0.933	0.992	0.991	0.986
FPR	0.992	0.992	0.992	0.992	0.662	0.992	0.686	0.736
AUC	0.993	0.991	0.987	0.996	0.954	0.997	0.996	0.985

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk and italics

indicate that the SVM classifier achieved the highest performance with 99.18% accuracy. However, the findings demonstrate that the NB classifier exhibited the worst classification accuracy performance with 93.33% accuracy.

We performed a statistical performance analysis of the algorithms on the Drupal 6.0.0 version dataset in experiment 3. In Table 5, the highest accuracies (85.7%) and (85.3%) were given for the LLTH-GRU and LLTH-LSTM algorithms, respectively. The results indicate that the classification accuracy increased by about 13.2% for the LLTH-GRU algorithm and 12.8% for the LLTH-LSTM algorithm versus AIS-based classifiers. Furthermore, the LLTH-GRU algorithm performed (TPR: 0.857, FPR: 0.742) better than the LLTH-LSTM (TPR: 0.855, FPR: 0.75) algorithm. The Immunos-99 algorithm obtained the worst accuracy performance with 44.3% accuracy. The CSCA and PAIRS algorithms exhibited comparable classification performance with 72.3% and 72.5% accuracy, respectively.

As shown in Table 6, the XGBoost classifier achieved the best classification performance with 81.21% accuracy. The MLP classifier obtained the worst classification performance with 72.77% accuracy. It can be observed that machine learning-based techniques result in poor

Table 5 Performance of AIS-based classifiers versus suggested algorithms in experiment 3

Statistical metrics	AIS-based classifiers							Suggested algorithms	
	CSCA	Immunos-1	Immunos-2	Immunos-99	AIRS1	AIRS2	PAIRS	LLTH-GRU	LLTH-LSTM
P	0.717	0.729	0.721	0.721	0.696	0.669	0.715	0.865	0.864
R	0.724	0.45	0.592	0.442	0.669	0.696	0.726	0.857	0.853
F	0.718	0.413	0.582	0.402	0.676	0.697	0.718	0.86	0.857
Accuracy	72.3	44.9	58.2	*44.3**	66.9	69.6	72.5	**85.7**	85.3
TPR	0.724	0.45	0.582	0.442	0.669	0.551	0.726	0.857	0.855
FPR	0.406	0.278	0.284	0.283	0.378	0.402	0.414	0.742	0.75
AUC	0.778	0.52	0.635	0.534	0.686	0.713	0.748	0.887	0.875

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk and italics

Table 6 Performance of standard classifiers in experiment 3

Statistical metrics	Standard classifiers						XGBoost	AdaBoost
	MLP	RF	k-NN	SVM	NB	J48		
P	0.708	0.759	0.734	0.746	0.739	0.754	0.793	0.775
R	0.728	0.767	0.738	0.748	0.752	0.748	0.783	0.736
F	0.705	0.760	0.736	0.705	0.733	0.75	0.765	0.753
Accuracy	*72.77**	76.73	73.76	74.75	75.24	75.7	**81.21**	79.2
TPR	0.728	0.767	0.738	0.748	0.752	0.748	0.812	0.792
FPR	0.480	0.364	0.368	0.516	0.442	0.318	0.653	0.713
AUC	0.763	0.789	0.745	0.765	0.778	0.769	0.846	0.824

The best accuracy results for the tested methods are written in bold, and the worst accuracy results are represented by using the asterisk and italics

performance when compared to the suggested algorithms. In general, the LLTH-GRU and LLTH-LSTM algorithms exhibited the highest accuracy performances, respectively, compared with AIS-based classifiers. The results showed the increased classification accuracies in all experiments achieved by the suggested algorithms. The results strongly recommend that the feature selection methods can leverage the performance of static analysis for vulnerability detection. For both the control classifier and the feature selection model, the random forest (RF) was used.

Figures 7, 8, 9, 10, 11, and 12 show the root-mean-square error (RMSE) performance results evaluated based on the RF classifier for the phpMyAdmin, Moodle, and Drupal datasets. We plotted the RMSE for traditional AIS-based classifiers and the suggested algorithms based on three experiments in all figures. We evaluated the impact of vulnerabilities through running tests ranging from 10 to 100. The figures summarized the root-mean-square error per 10 iteration number. Figure 7 demonstrates that the CSCA classifier achieved the best performance with 0.270 and the Immunos-2 classifier with a 0.297 error rate. As shown in Fig. 8, the suggested LLTH-GRU and LLTH-LSTM algorithms performed better when compared to any traditional AIS-based classifier with the root-mean-square error of 0.22 and 0.245, respectively, in experiment 1.

Table 7 shows the recent studies that have been conducted in vulnerabilities detection using deep learning technologies. We emphasize that LLTH-DNN can identify more vulnerabilities compared with other methodologies represented in Table 7. We can see that LLTH-GRU outperforms the other models in four recent studies conducted in vulnerabilities detection using deep learning. In all of the compared results, LLTH-LSTM and LLTH-GRU achieve statistical performance measures. In the [34, 40], 41, automatically learned features are more robust than in [7, 42, 43]. So, the proposed method got a promising result compared to other similar methods published methods.

According to Fig. 9, the CSCA and Immunos-2 classifiers achieved the best root-mean-square error of 0.98 and 0.1, respectively. In Fig. 9, it was observed that each of the suggested algorithms exhibited the increased performance of root-mean-square error with 0.0925 and 0.11 in experiment 2. The results indicated an improvement in the proposed algorithm's performance using the LLTH-based selection model for vulnerability detection. Likewise, we evaluated the root-mean-square error performance in experiment 3, and the obtained results are presented in

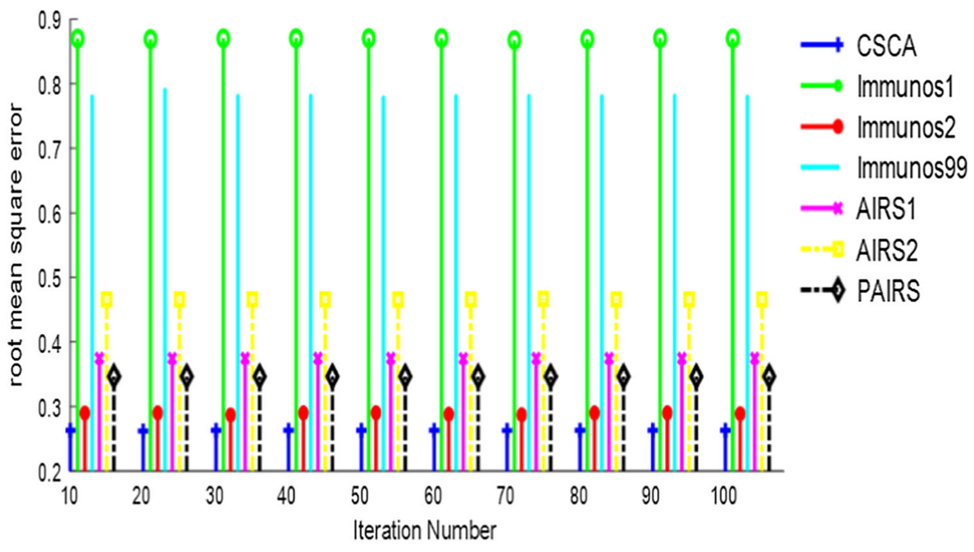


Fig. 7 RMSE performance of AIS-based classifiers in experiment 1

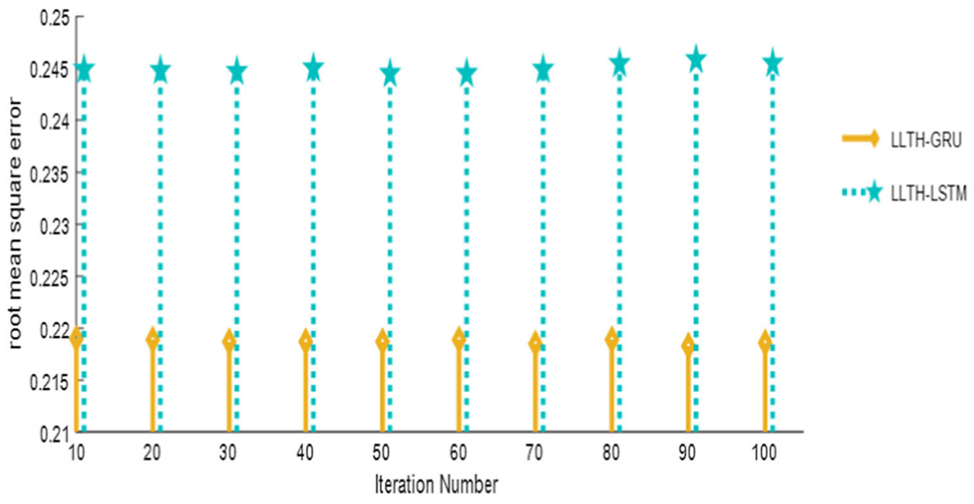


Fig. 8 RMSE performance of the suggested algorithms in experiment 1

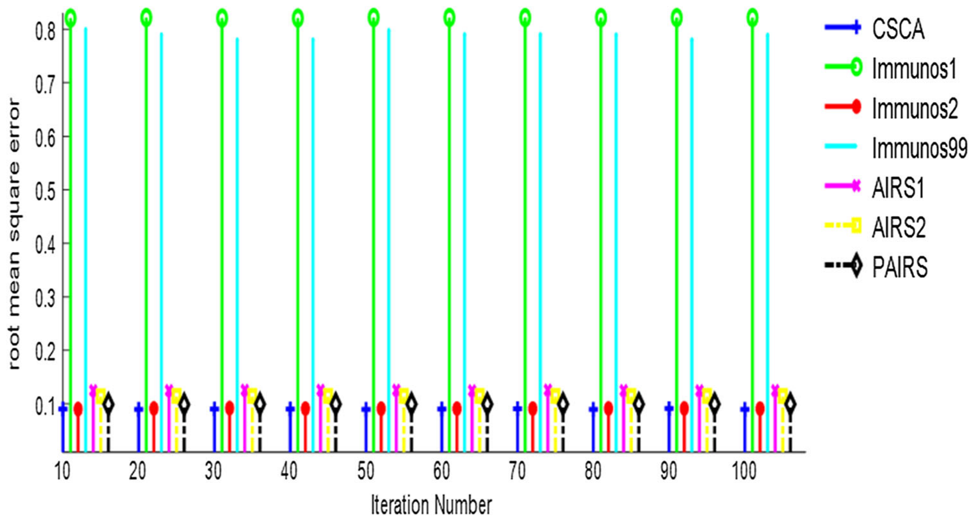


Fig. 9 RMSE performance of AIS-based classifiers in experiment 2

Figs. 11 and 12. According to Fig. 11, the best performance was achieved by the CSCA and PAIRS classifiers with an error of 0.520 and 0.525, respectively. Figure 12 summarizes the root-mean-square error performance of the

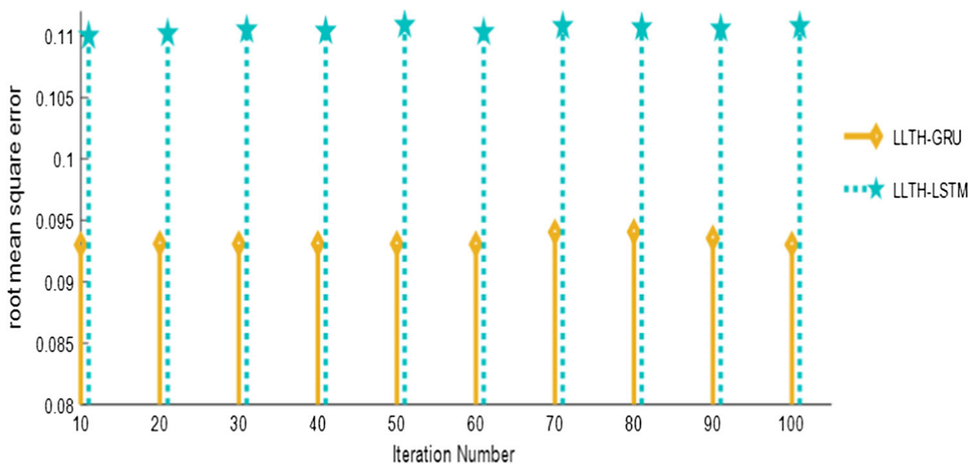


Fig. 10 RMSE performance of the suggested algorithms in experiment 2

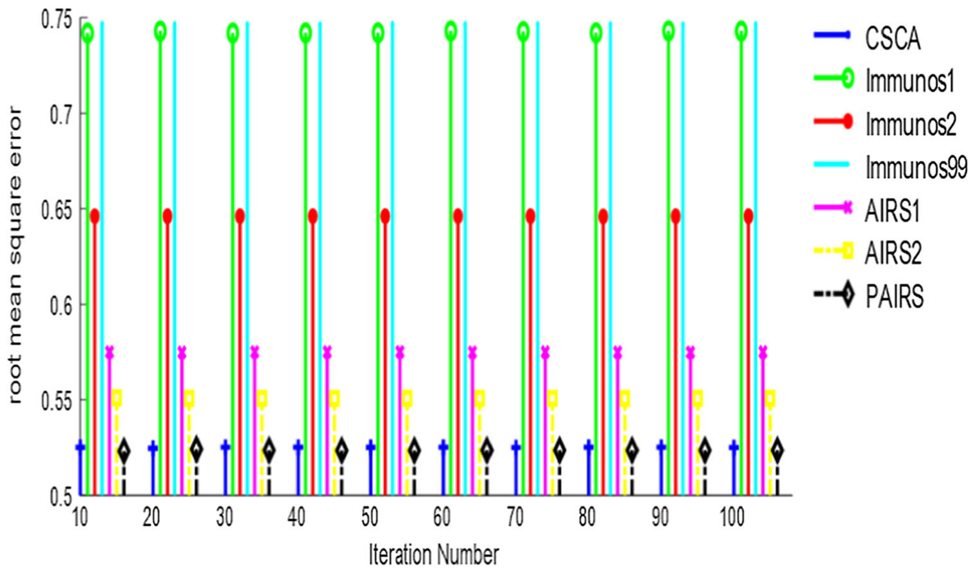


Fig. 11 RMSE performance of AIS-based classifiers in experiment 3

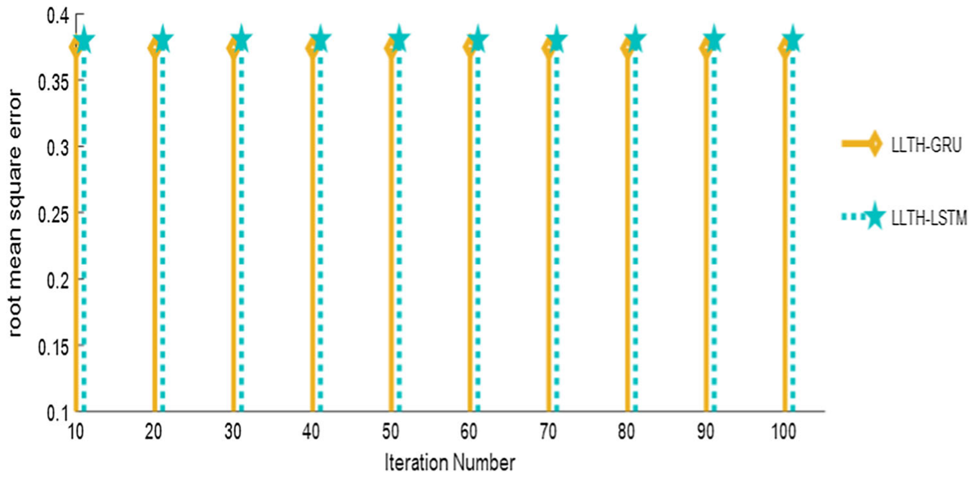


Fig. 12 RMSE performance of the suggested algorithms in experiment 3

suggested algorithms. The results show that the LLTH-GRU algorithm achieved the best root-mean-square error performance with an error rate of 0.370, and the LLTH-LSTM algorithm exhibited a comparable performance with an error rate of 0.387. We found that the suggested models,

Table 7 Recent studies conducted in vulnerabilities detection using deep learning

References	DNN model	Training data	Dataset	Reported results		
[1]	RNN	Automatically learned features	NVD, SARD	FPR: 5.7	P: 88.1	F1: 90.5
[39]	DNN	Automatically learned features	NVD, SARD	FPR: 1.4	P: 90.8	F1: 92.6
[40]	CNN, LSTM	Automatically learned features	SARD	FPR: 1.2	TPR: 87.6	F1: 91.1
[41]	CNN, RNN	Automatically learned features	NIST SATE IV	AUC: 95.4	P: 0.944	F1: 84
[42]	DNN, CNN	Automatically learned features	Open-source C-projects	–	P: 0.795	F1: 84.97
[43]	LSTM	Automatically learned features	Open-source Android-projects	R: 0.93	P: 0.92	F1: 0.91
Proposed LLTH-LSTM	GRU	Automatically learned features	PHPMyAdmin Dataset	FPR: 0.981	P: 0.984	F1: 0.986
Proposed LLTH-GRU	GRU	Automatically learned features	PHPMyAdmin Dataset	FPR: 0.992	P: 0.962	F1: 0.98

especially the LLTH-GRU algorithm, performed much better in experiment 2 than the other experiments.

6 Conclusion

This paper provided evidence that feature selection methodologies increase the classification accuracy and decrease the false-positive rate in the classification of static analysis. The proposed immunological feature selection methodology showed a statistically significant increase in the classification accuracy of vulnerabilities over a leveraging model feature. It is possible to employ the suggested model to select the relevant feature sets per project, programming language, or time frame. In summary, the feature selection model performed well in all the tests conducted. Using this model to classify vulnerabilities generated by the software, it is possible to increase their classifications. The performance of SVP models on the public phpMyAdmin, Moodle, and Drupal datasets was investigated. We compared standard AIS-based classifiers' performances and suggested models enhance the classification accuracy by using the proposed feature selection model. For future studies, text-based features acquired from the source code may be investigated. Furthermore, it is essential to assess the suggested methodology on other datasets applied with various programming languages to judge the models' applicability outside of the PHP programming language context.

Declarations

Conflict of interest There is no conflict of interest for authorship.

Ethical approval This manuscript does not contain any studies with human participants carried out by any of the authors.

References

1. Zou Q, Ni L, Zhang T, Wang Q (2015) Deep learning based feature selection for remote sensing scene classification. IEEE Geosci Remote Sens Lett 12(11):2321–2325
2. Alves H, Fonseca B, Antunes N (2016) Software metrics and security vulnerabilities: dataset and exploratory study. In: 12th European dependable computing conference (EDCC), Gothenburg, Sweden. pp 37–44
3. Williams L (2007) Toward the use of automated static analysis alerts for early identification of vulnerability-and attack-prone components. In: Second ınternational conference on ınternet monitoring and protection, San Jose, CA, USA. pp 18–18
4. Antonios G, Dimitris M, Diomidis S (2018) Vulinoss: a dataset of security vulnerabilities in open-source systems. In: Proceedings of the 15th ınternational conference on mining software repositories. ACM, pp 18–21
5. Koc U, Saadatpanah P, Foster JS, Porter A (2017) Learning a classifier for false positive error reports emitted by static code analysis tools. In: Proceedings of the 1st ACM SIGPLAN ınternational workshop on machine learning and programming languages, MAPL 2017. ACM , New York, NY, USA, pp 35–42
6. Twycross J, Aickelin U (2010) Information fusion in the immune system. Inf Fus 11(1):35–44
7. Li Z, Zou D, Xu S, Jin H et al (2018) VulDeePecker: A deep learning-based system for vulnerability detection, network and distributed systems security (NDSS) symposium 2018, San Diego, CA, USA ISBN: 1-1891562-49-5. http://dx.doi.org/https://doi.org/10.14722/ndss.2018.23158
8. Dolan-Gavitt B, Hulin P, Kirda E, Leek T, Mambretti A, Robertson WK, Ulrich F, Whelan R (2016) LAVA: large scale automated vulnerability addition. İn: IEEE symposium on security and privacy, SP 2016. pp 110–121. Doi: https://doi.org/10.1109/SP.2016.15
9. Goeschel K (2019) Feature set selection for ımproved classification of static analysis alerts, Nova Southeastern University, College of Computing and Engineering, CCE Theses and Dissertations
10. Fang Y, Han S, Huang C, Wu R (2019) TAP: A static analysis model for PHP vulnerabilities based on token and deep learning technology. PLoS ONE. https://doi.org/10.1371/journal.pone.0225196
11. Manjula C, Florence L (2019) Deep neural network based hybrid approach for software defect prediction using software metrics. Cluster Comput. https://doi.org/10.1007/s10586-018-1696-z

12. Kwon D, Kim H, Kim J et al (2019) A survey of deep learning-based network anomaly detection. Cluster Comput 22:949–961. https://doi.org/10.1007/s10586-017-1117-8
13. Abualigah L, Diabat A, Mirjalili S, Abd Elaziz M, Gandomi AH (2021) The arithmetic optimization algorithm. Comput Methods Appl Mech Eng 376:113609
14. Abualigah L, Yousri D, Abd Elaziz M, Ewees AA, Al-qaness MA, Gandomi AH (2021) Aquila optimizer: a novel meta-heuristic optimization algorithm. Comput Ind Eng. https://doi.org/10.1016/j.cie.2021.107250
15. Alnafessah A, Casale G (2020) Artificial neural networks based techniques for anomaly detection in Apache Spark. Cluster Comput. https://doi.org/10.1007/s10586-019-02998-y
16. Zlomislić V, Fertalj K, Sruk V (2017) Denial of service attacks, defences and research challenges. Cluster Comput 20:661–671. https://doi.org/10.1007/s10586-017-0730-x
17. Wang C, Yao H, Liu Z (2019) An efficient DDoS detection based on SU-Genetic feature selection. Cluster Comput 22:2505–2515. https://doi.org/10.1007/s10586-018-2275-z
18. Xue B, Zhang M, Browne WN, Yao X (2016) A survey on evolutionary computation approaches to feature selection. IEEE Trans Evol Comput 20(4):606–626
19. Zhang X, Liu F (2009) Feature selection based on clonal selection algorithm. Eval Appl. https://doi.org/10.4018/978-1-60566-310-4.ch009
20. Sharma A, Sharma D (2011) Clonal selection algorithm for classification. In: Liò P, Nicosia G, Stibor T (eds) Artificial immune systems. ICARIS 2011. Lecture notes in computer science, vol 6825. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22371-6_31
21. Ambusaidi M, He X, Nanda P, Tan Z (2016) Building an intrusion detection system using a filter-based feature selection algorithm. IEEE Trans Comput 65(10):2986–2998
22. Chess B, McGraw G (2004) Static analysis for security. IEEE Secur Priv 2(6):76–79. https://doi.org/10.1109/MSP.2004.111
23. Timmis J, Knight T, de Castro LN, Hart E (2004) An overview of artificial immune systems. In: Paton R, Bolouri H, Holcombe M, Parish JH, Tateson R (eds) Computation in cells and tissues. Natural computing series. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-06369-9_4
24. De Castro LN, Timmis JI (2003) Artificial immune systems as a novel soft computing paradigm. Soft Comput 7(8):526–544
25. Huang G, Li Y, Wang Q, Ren J, Cheng Y, Zhao X et al (2019) Automatic Classification method for software vulnerability based on Deep Neural Network. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2900462
26. Shin Y, Meneely A, Williams L, Osborne JA (2011) Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans Softw Eng 37(6):772–787
27. Abualigah L, Alsalibi B, Shehab M, Alshinwan M, Khasawneh AM, Alabool H (2020) A parallel hybrid krill herd algorithm for feature selection. Int J Mach Learn Cybern. https://doi.org/10.1007/s13042-020-01202-7
28. Archibald R, Fann G (2007) Feature selection and classification of hyperspectral images with support vector machines. IEEE Geosci Remote Sens Lett 4(4):674–677
29. https://www.evolvingsciences.com
30. Dudek G (2012) An artificial immune system for classification with local feature selection. IEEE Trans Evol Comput 16(6):847–860
31. Alom MDZ, Taha TM et al (2019) A state-of-the-art survey on deep learning theory and architectures. Electronics 8:292. https://doi.org/10.3390/electronics8030292
32. Goldberg MD, Qu Y, McMillin LM, Wolf W, Zhou L, Divakarla M (2003) AIRS near-real-time products and algorithms in support of operational numerical weather prediction. IEEE Trans Geosci Remote Sens 41(2):379–389
33. Sherstinsky A (2020) Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. Physica D 404:132306
34. Li Z, Zou D, Xu S, Jin H, Zhu Y, Chen Z (2018) SySeVR: A framework for using deep learning to detect software vulnerabilities. arXiv:1807.06756
35. Dasgupta D, Nino F (2009) Immunological computation: theory and applications. Taylor & Francis, London
36. Abualigah LM, Khader AT, Hanandeh ES (2018) Hybrid clustering analysis using improved krill herd algorithm. Appl Intell 48(11):4047–4071
37. Abualigah LM, Khader AT, Hanandeh ES, Gandomi AH (2017) A novel hybridization strategy for krill herd algorithm applied to clustering techniques. Appl Soft Comput 60:423–435
38. Abualigah L (2018) Feature selection and enhanced krill herd algorithm for text document clustering. Springer, Berlin . https://doi.org/10.1007/978-3-030-10674-4 **(ISBN: 1860-949X)**
39. Abualigah LM, Khader AT (2017) Unsupervised text feature selection technique based on hybrid particle swarm optimization algorithm with genetic operators for the text clustering. J Supercomput 73(11):4773–4795
40. Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, Ellingwood P, McConley M (2018) Automated vulnerability detection in source code using deep representation learning. İn: Proceedings of 17th IEEE ınternational conference on machine learning and applications (ICMLA). pp 757–762
41. Dam HK, Tran T, Pham T, Ng SW, Grundy J, Ghose A (2021) Automatic feature learning for predicting vulnerable software components. IEEE Trans Softw Eng 47(1):67–85. https://doi.org/10.1109/TSE.2018.2881961
42. Xiaomeng W, Tao Z, Runpu W, Wei X, Changyu H (2018) CPGVA: code property graph-based vulnerability analysis by deep learning. İn: Proceedings of the 2018 10th ınternational conference on advanced ınfocomm technology (ICAIT). IEEE, pp 184–188
43. Zhou Y, Liu S, Siow J, Du X, Liu Y (2019) Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Wallach H, Larochelle H, Beygelzimer A, d'AlcheBuc F, Fox E, Garnett R (eds) NIPS proceedings - advances in neural ınformation processing systems 32 (NIPS 2019) (Vol. 32). (Advances in Neural Information Processing Systems). Neural Information Processing Systems (NIPS)
44. Ghaffarian SM, Shahriari HR (2017) Software vulnerability analysis and discovery using machine-learning and data-mining techniques: a survey. ACM Comput Surv CSUR 50(4):1–36
45. Brucker AD, Deuster T (2014) U.S. Patent No. 8,881,293. Washington, DC: U.S. Patent and Trademark Office
46. Graves A. (2012) Long short-term memory. In: Supervised sequence labelling with recurrent neural networks. Studies in computational intelligence, vol 385. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24797-2_4
47. Chu Y et al (2019) DTI-CDF: a cascade deep forest model towards the prediction of drug-target interactions based on hybrid features. Brief Bioinform. https://doi.org/10.1093/bib/bbz152
48. Zhang YF et al (2020) SPVec: A Word2vec-ınspired feature representation method for drug-target ınteraction prediction. Front Chem 7:895
49. Wang X et al (2019) STS-NLSP: a network-based label space partition method for predicting the specificity of membrane transporter substrates using a hybrid feature of structural and semantic similarity. Front Bioeng Biotechnol 7:306. https://doi.org/10.3389/fbioe,2019.p.306-319

50. Junaid M et al (2019) Extraction of molecular features for the drug discovery targeting protein-protein interaction of Helicobacter pylori CagA and tumor suppressor protein ASSP2. Proteins: Structure, Function, and Bioinformatics. pp 837–849
51. Khan F et al (2020) Prediction of recombination spots using novel hybrid feature extraction method via deep learning approach. Front Genet 11:1052