Software Quality Journal (2025) 33:17
https://doi.org/10.1007/s11219-025-09714-7

RESEARCH

Improving software vulnerability severity prediction model performance with HDLN & FWFS: a two-stage feature selection approach

Ruchika Malhotra[1] · **Vidushi**[1,2]

Accepted: 3 February 2025

Abstract
Software Vulnerability Severity Prediction (SVSP) is an evolving domain of software engineering. Ensuring the robustness and continuous operation of software systems relies heavily on the accuracy of SVSP models. However, the effectiveness of SVSP models relies heavily on the features used for training. The presence of redundant, correlated, or irrelevant features can significantly degrade model performance. To address this issue, researchers employ Feature Selection (FS) methods to streamline feature sets, reducing computational costs in model development. However, selecting appropriate FS methods and utilizing their combined strength to achieve a more efficient model remains a challenge in many existing techniques documented in the literature. In this paper, to overcome the problems associated with a large number of features, two new FS modules are proposed which have not been explored so far. The first module employs a combination of Convolutional Neural Network and Bidirectional Long Short-Term Memory, referred to as the Hybrid Deep Learning Network (HDLN) module while the second module utilizes Information Gain and Grey Wolf Optimization techniques, forming the Filter Wrapper Feature Selection (FWFS) module. The efficacy of the proposed feature selection approach is evaluated through the Area under the Curve (AUC). Upon analyzing the results, it is determined that the proposed module enhances the performance of the SVSP model. The prediction models crafted using the HDLN module exhibited the most superior performance, boasting an average AUC value of 0.88. Following closely, the FWFS module, employing Extreme Gradient Boosting as a classifier, demonstrated the second-best performance, achieving an average AUC value of 0.86.

Keywords Hybrid Feature Selection · Prediction Model · Software Vulnerability · Machine learning algorithms

✉ Vidushi
vidushi_2k18phdco23@dtu.ac.in

[1] Department of Software Engineering, Delhi Technological University, New Delhi, India

[2] School of Engineering and Technology, Vivekananda Institute of Professional Studies-Technical Campus, New Delhi, India

1 Introduction

A software vulnerability is frequently exploited by attackers to gain access to a software system (Alves et al., 2016). The degree to which a software system is vulnerable considerably affects the system's overall quality (Erturk & Akcapinar, 2015). Moreover, with large software becoming a reality, the complexity and diversity grow exponentially, and it is impossible to develop error-free software. Therefore, developing a Software Vulnerability Severity Prediction Model (SVSPM) is crucial, which takes data available on software vulnerability as input. Software vulnerability reports, software metrics, and source code can be processed and can be taken as input to SVSPM. When textual data is used as input, high-dimensional features are likely to be developed. High-dimensional data suffer from data quality problems that degrade the model's performance and increase computational cost. To mitigate these problems, various FS methods (Liu et al., 2010) can be applied. FS methods select the most relevant features (variables) from a dataset, which improves model performance by reducing overfitting and computational cost. It helps enhance the model's accuracy and efficiency by removing irrelevant or redundant data and finding the optimal subset that holds maximum information. Previous research in this field indicates that if FS methods are not applied, the prediction model's performance will be degraded (Khazaei et al., 2016; Russell et al., 2018; Wang et al., 2019). Hence, the FS is considered a vital aspect of the prediction process. The FS methods can be broadly categorized into two categories (1) Filter methods, and (2) Wrapper methods (Liu & Yu, 2005). Researchers widely use filter methods because of their simplicity and speed. The choice of the classifier technique does not bias the result of the filter methods. When a learning technique is used in the selection process for the wrapper, they are computationally more expensive but perform better than filter methods.

In the past, research has been carried out on comparing the results of different FS methods and trying to find the best one (Jeon & Kim, 2021; Russell et al., 2018; Wang et al., 2019; Yong et al., 2020). Some studies (Stuckman et al., 2017) also demonstrated either no or slight improvement in the results using FS methods. Some studies (Davari et al., 2017; Kudjo et al., 2019; Tang et al., 2020) did not opt to go in this direction and applied only preprocessing techniques. Because of the variation in datasets in practical applications, no single FS method is best suited for all the datasets. Inspired by this, this study concentrates on developing a unified module for FS to eliminate the limitations found in existing FS methods. We can assume following scenario: A software company, responsible for the security of Mozilla products, wants to predict the likelihood of vulnerabilities in its software releases. The vulnerability data they possess is large and complex, including textual descriptions of bugs and reports. Predicting severity of vulnerabilities accurately is challenging because the dataset is highly imbalanced, and many features (words, phrases, and attributes) may be irrelevant, leading to dimensionality concerns. Reducing these irrelevant features while maintaining meaningful information is crucial for efficient and accurate predictions. To achieves this, in this study, by amalgamating a variety of FS techniques into a single module. This integration aims to mitigate the weaknesses inherent in individual FS methods, thereby enhancing the overall efficacy of feature selection processes. Through this approach, the framework offers a more comprehensive solution for FS. To this end, we have developed an SVSPM which automates the process of predicting the severity of software vulnerability. This helps speed up the process of prioritizing the vulnerabilities so that more severe ones can be

fixed first. When high dimensional data act as input to SVSPM, removing the curse of dimensionality becomes important. Therefore, two new FS modules are designed for dimensionality reduction.

The first module is based on deep learning methods. In literature, it is found that deep learning methods are gaining popularity and giving good results in this field (Das et al., 2021; Failed, 2021; Sun et al., 2023, 2024). The module uses the Convolutional Neural Network (CNN) and Bidirectional Long Short-Term Memory (Bi-LSTM) and is named as Hybrid Deep Learning Network (HDLN) module. In the HDLN module, the CNN model is a kind of deep neural network that is applied to many fields such as image classification, speech recognition, and bioinformatics. Convolution, pooling, and fully connected layers make up the structure of a standard CNN. Recently, CNN has been used for Natural Language Processing (NLP) problems and has given good results (Du et al., 2017; Ghourabi et al., 2020; Rhanoui et al., 2019). CNN can extract essential features from textual data. However, it cannot interlink the present information with the past information. For this, another deep neural network model, Bi-LSTM, is used in Schuster and Paliwal (1997) which is a kind of recurrent neural network that can do data processing in two directions containing two hidden layers, thereby giving better results in NLP (Tai et al., 2015; Yin et al., 2017).

The second module uses Information Gain (IG) and Grey Wolf Optimization (GWO). It is named as Filter-Wrapper FS (FWFS) module. In the FWFS module, Information Gain is used as the term frequency-inverse document frequency, which is also used to find class labels for term-weighting, similar to FS. Among metaheuristic optimization techniques, Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), and Genetic Algorithm (GA) are popular ones that are used to find optimal features. However, it has some drawbacks, such as being trapped in local optima (Xue et al., 2016). A new optimization technique called Grey Wolf Optimizer (GWO) was created by Mirjalili et al. (2014), and it has recently received a lot of attention for addressing the problem of FS in various disciplines. It tries to copy the hunting behavior of wolves and gives much robustness compared to PSO and GA optimizers against initialization (Emary et al., 2015). On the bright side, GWO is known for its fast convergence rate and computational efficiency, particularly for small to medium-sized optimization problems. It can quickly explore the search space and converge to a solution within a reasonable amount of time, making it suitable for our problem. Moreover, GWO has shown promising performance in some real-world optimization problems. IG is a filter method and GWO is a wrapper method. Filter methods are considered to be fast whereas wrapper methods are considered to be more accurate. Therefore, IG acts as a Fast Feature Selector (FFS), and GWO acts as an Accurate Feature Selector (AFS). Combining these two in the FWFS module also combines the strengths of both aiming to eliminate the non-significant features and find an optimal subset of features for SVSP. As FS has to be applied to textual data of large dimensions, FFS is applied first and IG score is found for all the features. Features are selected according to their increasing order of IG scores. A high value of Information Gain indicates that the feature provides a significant amount of information about the target variable and is therefore valuable for prediction or classification tasks. After the selection of features via FFS, AFS is applied in order to find the optimal subset of features. As FS is a binary problem, standard GWO is converted to binary GWO (bGWO) which updates the grey wolf position vector to be binary (Emary et al., 2015).

To evaluate the performance of the SVSPM, Area under the Curve (AUC) values are used as a performance parameter across five Mozilla datasets. The values of AUC are recorded for all four levels of severity. The choice of AUC is made as it is considered

a good choice among various other parameters as it finds a balance between specificity and sensitivity values. it is also considered to be a nice option when dealing with imbalanced data problems (He & Garcia, 2009). Statistical testing is also applied to the result values obtained to find a significant difference among various experimental scenarios, judging the effectiveness of the proposed modules for FS.

The primary contributions of this study are outlined as follows:

1. This paper addresses vulnerability data presented in textual format and tackles the challenges posed to prediction models by dimensionality concerns.
2. The paper introduces two novel FS modules. The HDLN module explores the efficacy of integrating CNN and the Bi-LSTM network, while the FWFS module investigates the effectiveness of combining the metaheuristic algorithm GWO with IG.
3. Extensive testing is conducted to evaluate the functionality of the proposed strategy in terms of model performance, aiming to assess the effectiveness of the suggested FS modules.
4. Statistical analysis is employed to compare the performance of models utilizing the proposed FS modules, aiming to identify any significant differences.

In this study, the following research questions are addressed:

RQ1. How does SVSPM perform better when different FS methods are used to make the data less dimensional?

The effectiveness of various methods used for dimensionality reduction is measured on the dataset of five different products of Mozilla using AUC as a performance measure.

RQ1.1 When the HDLN module is used to lower the dimensionality of the data, what performance improvement does SVSPM experience?

With the aid of this RQ, we looked into our findings after implementing the deep learning FS methods and discovered enhancements in the SVSPM's performance when compared to the performance of the SVSPM without the FS method.

RQ1.2 When the FWFS module is used to lower the dimensionality of the data, what performance improvement does SVSPM experience?

With the aid of this RQ, we looked into our findings after implementing the filter-wrapper FS method and discovered enhancements in the SVSPM's performance when compared to the performance of the SVSPM without the FS method.

RQ2. Which module of FS among the two proposed modules gave the best results?

Addressing this RQ, the most effective method of FS is found, and a final judgment is given, stating the best method for FS.

RQ3. Whether the performance of SVSPM developed boosted significantly with the help of FS methods used to reduce the dimensionality of data?

Based on the results of the Friedman test, a significant difference in the performance of SVSPM is found.

This paper is organized as follows: Sect. 2 provides a brief review of past work. Section 3 outlines the preliminary steps required to conduct the study. The methodology for developing the proposed framework is detailed in Sect. 4. Section 5 presents the results obtained. Potential threats to validity are discussed in Sect. 6. Finally, Sect. 7 offers concluding remarks.

2 Related work

The SVSPM prediction model is primarily constructed using machine learning algorithms. To improve its performance, researchers combined feature selection (FS) techniques with particular classifiers. In the sections that follow, we will examine previous research on the development of SVSPM and delve into studies on feature selection using the hybridization of filter wrapper methods and deep learning methods.

2.1 Software vulnerability severity prediction

In this section, we have listed the latest research done on the development of SVSPM. The author in Kekül et al. (2021) proposes a machine learning framework to automate the classification of software vulnerabilities. It utilizes Natural Language Processing (NLP) techniques like bag of words, TF-IDF, and n-grams for feature extraction, and applies Naïve Bayes, decision trees, KNN, MLP, and random forest for prediction. The study evaluates the model using data from CVSS 2.0 and CVSS 3.1 and compares the effectiveness of different NLP techniques paired with various machine learning algorithms to determine the best combination for accurate vulnerability prediction.

The study (Chen et al., 2020) introduces a new approach term frequency-inverse gravity moment (TF-IGM) as an alternative to the commonly used term frequency-inverse document frequency (TF-IDF) for better weighting and identifying important terms in vulnerability descriptions. The study finds that TF-IGM outperforms the traditional TF-IDF in classifying software vulnerabilities. Additionally, the use of feature selection techniques significantly improves classification accuracy.

The study (Kudjo et al., 2020a) uses the concept of Bellwether analysis in predicting software vulnerability severity. The Bellwether analysis identifies a "lead" dataset (the bellwether) that performs well in forecasting for other similar datasets. This method can improve the accuracy of vulnerability severity predictions by leveraging transfer learning techniques.

The authors in Sharma et al. (2021) propose a method for prioritizing software vulnerabilities based solely on their descriptions. The authors eliminate the need for traditional scoring systems by using Natural Language Processing (NLP) techniques like word embeddings combined with a convolutional neural network (CNN). This approach helps categorize vulnerabilities into different severity levels, which improves efficiency in prioritization.

Recent studies focus on predicting vulnerability severity, with an emphasis on effectively handling textual data from vulnerability descriptions. Along these lines, our work also addresses the core challenge of converting textual data into meaningful feature vectors while overcoming the curse of dimensionality. To tackle this, we developed two advanced modules: the first leverages deep learning techniques, and the second utilizes evolutionary computing, providing a more efficient approach to feature selection and vulnerability prediction.

2.2 Feature selection using hybrid filter wrapper algorithm

Alomari et al. (Alomari et al., 2022) study proposes a hybrid feature selection method combining Principal Component Analysis (PCA) and the Grey Wolf Optimizer (GWO) algorithm for Arabic news article classification. This approach enhances classification

accuracy by reducing the feature space effectively. Yang et al. (Yang et al., 2024) propose a fast dual-module hybrid feature selection algorithm designed for high-dimensional data. The method integrates a filter-based module for rapid feature ranking and a wrapper-based module to refine the selected features. By combining these two approaches, the algorithm effectively reduces computational complexity while maintaining high classification accuracy. The study demonstrates the algorithm's efficiency and performance across various high-dimensional datasets, making it suitable for large-scale feature selection tasks. Robindro et al. (Robindro et al., 2024) introduce a hybrid distributed feature selection method combining Particle Swarm Optimization (PSO) with Mutual Information (MI). The MI filter ranks features, and PSO refines the selection by searching for optimal feature subsets. This approach enhances feature selection efficiency and improves classification accuracy in distributed environments. Hancer et al. (Hancer, 2024) introduce a hybrid wrapper-filter feature selection method, enhanced by an improved evolutionary algorithm and a novel initialization scheme. This approach balances computational efficiency and accuracy, accelerating the search for optimal feature subsets. The method outperforms traditional techniques in high-dimensional datasets and improves classification accuracy. Anju et al. (Anju & Judith, 2024) present a hybrid feature selection method for predicting software defects, combining both wrapper and filter approaches to enhance prediction accuracy. The method selects the most relevant features to improve model performance while reducing computational complexity. It demonstrates superior results in identifying software defects compared to traditional feature selection techniques.

2.3 Feature selection using hybrid deep learning algorithm

Jain et al. (Jain et al., 2020) present a sarcasm detection model for mash-up languages using a soft-attention-based bi-directional LSTM and a feature-rich CNN. The bi-directional LSTM captures contextual dependencies from both past and future tokens, while the CNN extracts critical features like syntactic and semantic cues. This hybrid approach significantly improves sarcasm detection in multilingual social media content. Ayetiran et al. (Ayetiran, 2022) proposes a hybrid CNN-BiLSTM model with an attention mechanism for aspect-based sentiment classification. The model captures both local features and long-range dependencies in text, while the attention mechanism enhances sentiment prediction by focusing on important words. This approach improves accuracy, particularly when applied to benchmark datasets. The paper (Thekkekara et al., 2024) presents an attention-based CNN-BiLSTM model for detecting depression in social media text. The CNN captures local patterns in text, while the BiLSTM handles long-term dependencies, and the attention mechanism highlights key depression-related words. This approach improves the accuracy of detecting depressive content in user posts on social platforms. The work in Sun and Chu (2020) presents a hybrid model combining Convolutional Neural Networks (CNN), Bidirectional Long Short-Term Memory (BiLSTM), and an attention mechanism for text sentiment analysis. CNN captures local features, while BiLSTM processes long-range dependencies, and attention enhances important feature focus. The model achieves improved accuracy in sentiment classification tasks compared to traditional methods. The effectiveness of the CNN-BiLSTM hybrid model in sentiment analysis encouraged us to adapt it to our own field of study aiming to assess its impact on the performance of the model, expecting similar improvements.

The above-stated literature depicts the success of these hybrid approaches. Nonetheless, the "No Free Lunch" (NFL) theorem in optimization (Wolpert & Macready, 1997) proves

logically that, with so many problems out in the research domain, no single optimization algorithm can solve all such problems. Relating it to our context of FS, no algorithm can find the optimal set of features for every dataset. Because of this, attempted to hybridize and improve two of the most recent and popular algorithms, GWO and WOA, to solve binary FS problems over textual data more effectively.

3 Preliminaries

In this section of the paper, various preliminaries required for this study are summarized.

3.1 Independent and dependent variables

This study aims at the severity prediction of security vulnerabilities, which is a multi-class problem. The output label or dependent variable is the severity level of vulnerabilities, which can take four values, 'low,' 'medium,' 'high,' and 'critical'. A dataset of vulnerability reports of different software is collected to make this prediction. Several fields are present in a vulnerability report, for example, CVE ID, vulnerability type, publish date, update date, access, complexity, authentication, etc. However, we used an unstructured portion of the vulnerability report for making a prediction, i.e., vulnerability description. Therefore, vulnerability description acts as an independent variable of our study.

3.2 Data collection

Vulnerability reports of five different products of Mozilla are used as datasets in this study. These are Mozilla Firefox (MF), Mozilla Seamonkey (MS), Mozilla Thunderbird (MT), Mozilla Firefox ESR (MF ESR), and Mozilla Thunderbird ESR (MT ESR). The vulnerability reports are extracted which are offered by the CVE following the CVSS framework (Zerkane et al., 2016) for scoring the severity of the vulnerability. This scoring system gives a score to vulnerability in the range of 0 to 10. According to version v3.0 of CVSS, which is the most recent version, a 'None' rating is given to vulnerabilities with a 0.0 score, a 'Low' rating is given to vulnerabilities with a score of 0.1–3.9, 'Medium' rating is given to vulnerabilities with a score of 4–6.9, 'High' rating is given to vulnerabilities with a score of 7–8.9 and 'Critical' rating is given to vulnerabilities with a score of 9–10. Table 1 below gives the dataset's details in context with severity labels. Out of all the attributes of a vulnerability report, vulnerability description is used to carry out experiments in this study as it is unstructured data. Mining unstructured data is a challenge as it is vague and may contain noise. Figure 1 gives

Table 1 Dataset Description

Dataset	Low	Medium	High	Critical
MF	164	897	296	516
MT	22	372	164	363
MS	13	291	61	333
MF ESR	10	284	175	203
MT ESR	1	69	15	131

an example of a vulnerability and all the attributes present in the vulnerability report related to that vulnerability.

3.3 Data preprocessing

Before working with the data, it needs to be pre-processed so that it becomes suitable to be given as input to the classifier using ML algorithms. In our case, we have unstructured data. This unstructured data is pre-processed using standard techniques: tokenization, stop word removal, and stemming (Vijayarani et al., 2015). In tokenization, a sequence of words is divided into individual, meaningful words termed tokens. In stop word removal, various words that do not hold any important information and act as connectors helping with framing the sentences are removed from the text. Stemming helps in reducing any word to its root form, thus reducing duplication.

3.4 Feature selection methods

3.4.1 Information gain

Among filter methods for FS, we used Information Gain (IG) in this study. It is used vividly in the literature. However, its limited use for mining vulnerability reports (Khazaei et al., 2016; Kudjo et al., 2020a; Malhotra & Vidushi, 2021) made us explore it for mining vulnerability reports.

IG (Schuster & Paliwal, 1997) just identifies the words that best clarify the desired concept. It determines the total amount of bits needed to encode any class distribution L is G (L), as seen below.

$$G(L) = -\sum_{c \in C} \frac{n(l)}{N} log_2 p(l) \tag{1}$$

$$\text{where}, N = \sum_{c \in C} n(l) \tag{2}$$

After observing an attribute, A, the number of bits needed to encode a class will be:

$$G(L|A) = -\sum_{a \in A} p(a) \sum_{l \in L} p(a) log_2(p(a)) \tag{3}$$

The attribute having the highest IG is given the highest rank as a high value of Information Gain denotes that a feature is useful for prediction since it offers important information about the target variable, which is denoted by the symbol Ai

$$Infogain(Ai) = \mathrm{G(L)} - \mathrm{G}\left(\frac{\mathrm{L}}{\mathrm{Ai}}\right) \tag{4}$$

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2021-38502			Exec Code	2021-11-03	2022-07-12	4.3	None	Remote	Medium	Not required	Partial	None	None

Thunderbird ignored the configuration to require STARTTLS security for an SMTP connection. A MITM could perform a downgrade attack to intercept transmitted messages, or could take control of the authenticated session to execute SMTP commands chosen by the MITM. If an unprotected authentication method was configured, the MITM could obtain the authentication credentials, too. This vulnerability affects Thunderbird < 91.2.

Fig. 1 Example of vulnerability report and attribute present related to a vulnerability

The higher value of IG indicates more significance of that term. Hence top N terms having the top values of IG can be selected to construct the model.

3.4.2 Grey wolf optimization

The evolutionary algorithm Grey Wolf Optimization (GWO) has recently been released. Mirjalili et al. (Mirjalili et al., 2014) have proposed it. Based on the social structure and hunting habits of grey wolves, this algorithm's foundation was created. Alpha, beta, delta, and omega are the four hierarchy levels that the GWO algorithm assigns to wolves. The alpha group represents the best option, followed by the beta group for the runner-up position, and the delta group for the third-best option. The remainder of the answers fall into the category of omega. Encircling the prey, stalking the prey, and finally attacking the prey are the three steps that make up the GWO approach. The first stage of encircling the prey can be depicted mathematically using the following equation:

$$\vec{Y}(i+1) = \vec{Y}_t(i) + \vec{P}.\vec{R} \tag{5}$$

$$\vec{R} = \left|\vec{Q}.\vec{Y}_t(i) - \vec{Y}(\mathrm{i})\right| \tag{6}$$

where $\vec{Y}$ and $\vec{Y}_t$ indicate the position of the wolf vector and the prey vector at an iteration i. Coefficients $\vec{P}$ and $\vec{Q}$ are calculated as depicted below:

$$\vec{P} = 2\vec{p}.\vec{n}_1 - \vec{p} \tag{7}$$

$$\vec{Q} = 2.\vec{n_2} \tag{8}$$

$\vec{n}_1$ and $\vec{n}_2$ are vectors that were produced at random in the range [0, 1], and $\vec{p}$ decreases linearly from 2 to 0 over iterations. In the second step or hunting, alpha signifies the best probable outcome, while beta and delta indicate potential prey locations. The best places available in the decision space are updated by Omega based on these three best options. This method of hunting is stated as follows:

$$\vec{Y}(i+1) = \frac{y_1 + y_2 + y_3}{3} \tag{9}$$

where $\vec{y_1}, \vec{y_2}, \vec{y_3}$ are the three best solutions at iteration i. These are calculated as follows:

$$\vec{y_1} = \vec{Y_\alpha} - P_1.\left(\vec{R_\alpha}\right) \tag{10}$$

$$\vec{y_2} = \vec{Y_\beta} - P_2.\left(\vec{R_\beta}\right) \tag{11}$$

$$\vec{y_3} = \vec{Y_\delta} - P_3.\left(\vec{R_\beta}\right) \tag{12}$$

and $\vec{R_\alpha}$, $\vec{R_\beta}$ and $\vec{R_\delta}$ are calculated as follows:

$$\overrightarrow{R_\alpha} = \left|\overrightarrow{Q_1}.\overrightarrow{P_\alpha}. - \vec{P}\right| \tag{13}$$

$$\overrightarrow{R_\beta} = |\overrightarrow{Q_2}.\overrightarrow{P_\beta}. - \vec{P}| \tag{14}$$

$$\overrightarrow{R_\delta} = |\overrightarrow{Q_3}.\overrightarrow{P_\delta}. - \vec{P}| \tag{15}$$

The attacking mechanism of grey wolf is expressed on the basis of vector $\vec{p}$. The range of this vector is [−p, p]. Over iterations, the value of $\vec{p}$ is lowered linearly from 2 to 0 and is denoted by:

$$\vec{p} = 2 - i.\frac{2}{MIter} \tag{16}$$

where MIter stands for the maximum iterations, and i represents iteration counter Fig. 2.

In the binary version of GWO (Mirjalili et al., 2014), to deal with binary optimization problems such as the FS problem (Firpi & Goodman, 2004), the continuous search space is converted to binary using the sigmoid transfer function using Eq. (17).

$$x_{sig} = \frac{1}{1 + e^{-x_i}}, \tag{17}$$

$$x_{binary} = \{0 if rand < x_{sig} 1 otherwise \tag{18}$$

where x_{sig} is a continuous feature value that is inputted to the sigmoid function, rand is a random number generated from the uniform distribution that belongs to [0,1], and x_{binary}, is the updated binary position.

3.5 Deep learning models

3.5.1 Basic CNN model

The CNN model is a deep neural network used in a variety of industries, including bioinformatics, image classification, and speech recognition. Layers with convolution, pooling, and fully linked connections make up the conventional CNN design. CNN has recently been employed to solve NLP issues and has produced successful outcomes (Du et al., 2017; Ghourabi et al., 2020; Rhanoui et al., 2019). Apart from that (Kim & Jeong, 2019; Li et al., 2021; Schirrmeister et al., 2017; Sharma et al., 2021) provide a strong foundation for the claim that CNNs are effective in extracting features from text making the use of the max-pooling layer. For example, through the use of max pooling with a 2×2 window and stride of 2, we efficiently reduce the size of the data, retaining only the most critical information

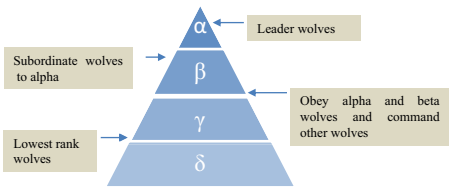

Fig. 2 Hierarchy of wolves

from the text without losing essential n-grams. This allows us to handle the large feature space typical in textual data.

For example, Let's assume we have a small grid of values representing n-gram scores like this:

[5, 1, 2, 4]
[9, 6, 3, 8]
[1, 2, 4, 0]
[7, 8, 1, 5]

After applying a 2×2 max pooling layer with a stride of 2, the result would be:

[9, 8]
[8, 5]

This reduces the overall data size by half but keeps the most important values (the maximum ones).

The polling layer of the CNN model lessens the computational complexity. The output size of one stack layer to the next is decreased by the polling approach while vital data is kept intact. The most popular polling method is known as max-polling, which places the max value element in the polling window. The output of the pooling layer is fed into and mapped to the next layers by the flattening layer. The CNN model's final layer is the fully linked layer.

3.5.2 Bidirectional-long short-term memory networks

CNN can extract important features from textual data. However, it cannot interlink the present information with the past information. While CNNs are powerful for pattern recognition tasks, they may face challenges in directly capturing temporal dependencies. Addressing this limitation often requires incorporating additional architectures or techniques designed specifically for handling sequential data. For this, another deep neural network model, Bi-LSTM can be used. Bi-LSTM (Schuster & Paliwal, 1997) is a type of LSTM model. It belongs to the category of RNN. It processes the data in both forward and backward directions of data in both directions making Bi-LSTM different from the LSTM network. BiLSTMs handle sequential data by processing the input sequence in both forward and backward directions simultaneously, allowing them to capture information from past and future contexts and model long-range dependencies effectively. This makes them well-suited for a wide range of sequential data tasks where understanding the context from both directions is important. Additionally, it has produced positive outcomes in natural language processing (Tai et al., 2015; Yin et al., 2017). This methodology has proven to be effective, particularly when analyzing documents and extensive entries. It creates a connection between the words in the document to forecast its tone.

3.6 Data balancing strategy

Most of the ML algorithms are designed on the assumption that the data over which they will be working is balanced. If applied to imbalanced data, the results often become skewed, showing biased outcomes toward the majority class (Kekül et al., 2021). As shown in Table 1, the data in this case is imbalanced, with low and high classes in the minority for all five databases, whereas medium and critical classes dominate. To address this,

data balancing techniques like oversampling and undersampling are often employed. These methods aim to improve the model's ability to handle class imbalance by either increasing the number of minority class instances or reducing the majority class.

One of the advantages of data balancing techniques is that they help improve model performance by ensuring that minority classes are better represented, thus enabling the model to capture more meaningful patterns across all classes. Balancing techniques also help to reduce bias in the model's predictions, leading to a more equitable evaluation of all classes. This is particularly important in real-world applications, where imbalanced data is common, such as in fraud detection, medical diagnosis, or rare event prediction.

However, data balancing methods come with disadvantages as well. Oversampling techniques, for example, can increase the risk of overfitting by simply duplicating minority class instances or generating synthetic ones, which may cause the model to learn noise rather than general patterns. Undersampling, on the other hand, reduces the amount of data from the majority class, which can result in a loss of valuable information and lead to poorer overall model performance. Additionally, both techniques may introduce longer training times, as oversampling increases the dataset size while undersampling reduces the diversity of the data available for learning.

In this work, SMOTE (Synthetic Minority Oversampling Technique) is employed to address the problem of data imbalance, which offers a solution by generating synthetic samples of the minority class through interpolation rather than duplication (Spelmen & Porkodi, 2018). While it mitigates some of the challenges of traditional oversampling, the broader advantages and disadvantages of data balancing techniques remain important considerations in model development.

3.7 Classification algorithms

Under this category, we concentrated on ensemble learners. It combines multiple ML algorithms to achieve good results to make a single prediction model. The main objective here is to compensate for the error caused due to one ML algorithm with the help of another ML algorithm and get a better-performing model overall (Sagi & Rokach, 2018). Ensemble classifiers often outperform individual base learners, especially in terms of predictive accuracy. By combining multiple models, ensemble classifiers can capture diverse patterns in the data, leading to more robust and accurate predictions. Their effectiveness in improving predictive accuracy and generalization performance has been well-established in both research and practice. This study used three ensemble learners: Bagging, Gradient Boosted Decision Tree (GBDT), and eXtreme Gradient Boosting (XGBoost). Bagging (bootstrap aggregation) is an independent method. Its pseudo-code can be referred from Breiman et al. (Breiman, 1996). It creates a composite classifier by combining the output of different learning algorithms into a single prediction model. On the other hand, unlike bagging, boosting the performance of the next learning technique depends on the prior ones that are developed. More details on GBDT and XGBoost can be found in Bahad and Saxena (2020).

3.8 Performance measures

The base of any performance measure is the confusion matrix. With the help of the confusion matrix, further parameters of performance measures can be calculated. In this study, the problem is not just a binary problem. It is a multiclass classification problem.

In this study, we used AUC as the performance measure to evaluate the effectiveness of our experiment conducted. AUC as a performance measure is a popular choice among researchers as it finds a balance between both specificity and sensitivity values. Additionally, AUC is insensitive to changes in the class distribution or prevalence of the positive class (He & Garcia, 2009). Overall, AUC offers a more comprehensive and robust evaluation of classifier performance compared to traditional metrics like accuracy, sensitivity, specificity, and precision, making it widely used in classification tasks. Its value ranges from 0 to 1 and is shown by a plot of the sensitivity and specificity values on the x- and y-axes, respectively. If the value of AUC is substantial and reaches 1, the prediction model is regarded as having produced accurate results. An AUC value of 1 denotes the model's perfect performance; an AUC value between 0.9 and 0.99 denotes the model's excellent performance; an AUC value between 0.8 and 0.89 denotes the model's good performance; an AUC value between 0.7 and 0.79 denotes the model's fair performance; and an AUC value below 0.7 denotes the model's subpar performance.

3.9 Statistical tests

According to the experiment, appropriate statistical tests are found to perform hypothesis testing. It is necessary to apply statistical tests as it can be misleading to derive a conclusion based on performance measures (Olson & Delen, 2008). Statistical tests are categorized as parametric and nonparametric statistical tests. Parametric statistical tests are used if the data on which it is needed to be applied follows some strict assumptions. If not, then nonparametric statistical tests are applied. The selection of statistical tests also depends on the number of data samples we deal with. Are we going to apply the test to two or more two samples that are dependent or independent of each other. The researcher finds an appropriate statistical test applicable to his study based on all these conditions. One-way ANOVA, Mann Whitney U test, Friedman test, and Wilcoxon signed-rank test are some of the statistical tests. According to our data, the widely accepted Friedman test is applicable.

4 The proposed methodology

This section outlines the methodology proposed in the paper, which focuses on dimensionality reduction and the development of a feature selection (FS) module aimed at improving model performance. The primary objective is to introduce two novel FS modules not addressed in existing literature: one based on Filter-Wrapper FS (FWFS), and the other based on a Hybrid Deep Learning Network (HDLN). To evaluate the effectiveness of the proposed modules, five additional scenarios are considered. These scenarios involve developing SVSPMs using different FS methods: one without FS, one utilizing Information Gain (IG) for FS, and another employing Grey Wolf Optimization (GWO) for FS. The rationale behind selecting IG and GWO for FS is to isolate individual components of the FWFS module and assess their impact on results. Furthermore, Convolutional Neural Network (CNN) and Bidirectional Long Short-Term Memory (Bi-LSTM) models are used for FS in two additional scenarios. These choices aim to investigate whether the combined use of CNN and Bi-LSTM, as in the HDLN module, leads to performance improvements.

4.1 Main scenario 1: HDLN module

This module integrates two neural networks, CNN and Bi-LSTM. This combination is chosen because CNN is renowned for its effectiveness in extracting features from text. Meanwhile, the Bi-LSTM network processes data in both forward and backward directions, thereby preserving the chronological order of the data. By integrating these architectures, the resulting model gains the ability to extract hierarchical representations that encompass both spatial and temporal aspects of the data. This fusion enhances the model's contextual understanding, robustness to variability, and effectiveness in capturing complex relationships. More specifically, CNN is effective in learning hierarchical representations of features from raw input data. CNNs possess the capability to automatically extract features at various levels of abstraction, effectively `performing feature selection by emphasizing the most discriminative features for the given task. However, they cannot establish connections between present and past data. Hence, adding Bi-LSTM addresses this limitation by simultaneously processing input sequences in both forward and backward directions, ensuring effective handling of sequential data. Consequently, this combined architecture proves highly suitable for a diverse range of sequential data tasks wherein understanding the context in both directions is essential.

The proposed framework comprises three sections as illustrated in Fig. 3. The number of layers and parameters employed by the HDLN module is depicted in Fig. 4. Further in the text, the structure of the HDLN module is delineated, along with a detailed description of its parameters.

Preprocessing: In this phase, raw data is preprocessed using tokenization, stop word removal, and stemming. The reduced data is then fed into the embedding layer using Word-2Vec embedding to produce an embedding matrix used as input by the convolution layer. The text is converted via Word2Vec into a vector of numerical values. The embedding matrix is composed of a sentencing matrix which may be written as:

$$S_m = \{V(W_1), V(W_2), \ldots\ldots, V(W_i)\} \tag{19}$$

where Wi is the word of the sentence, and V (W_i) is the word vector of dimension K. Maximum sequence length is 250, i.e., i is taken to be 250 here. K is the number of integers to which Word2Vec converts every word into. In this paper, the value of K is 100.

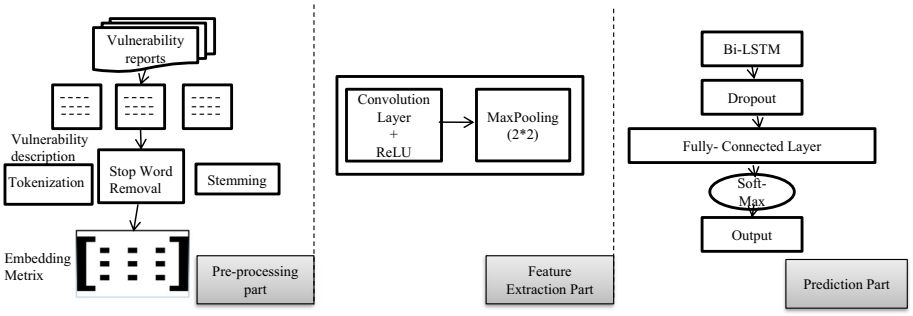

Fig. 3 Framework of HDLN module

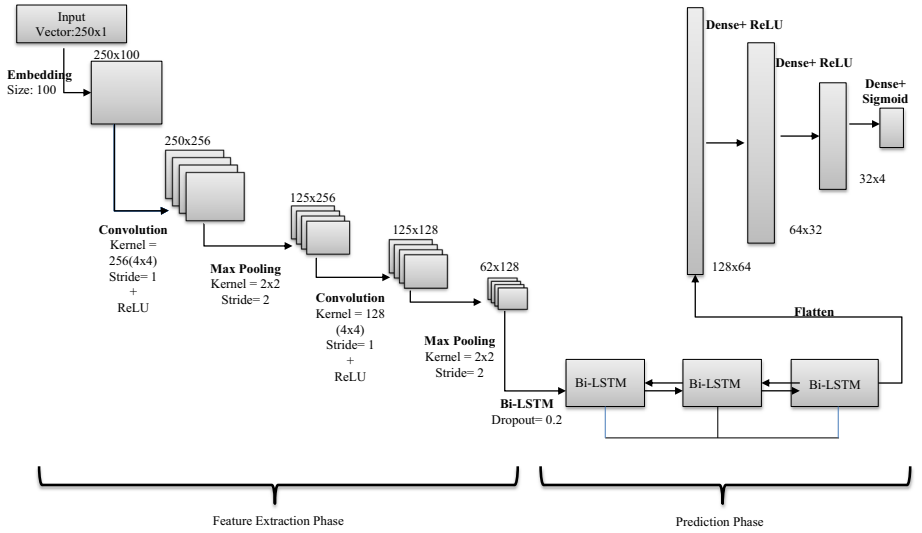

Fig. 4 Layers and parameters of NN of HDLN module

Feature Extraction based on CNN: The embedding matrix generated in the above step is taken as input. Two convolution layers and two max-pooling layers are applied alternatively in this stage. The first convolution layer contains 256 filters of dimension 4×4, and the second convolution layer contains 128 filters of dimension 4×4. Filters applied in the convolution layer also act as n-gram detectors. As the filter slides from top to bottom with a stride of 1 and passes through the whole S_m, it generates a local eigenvector set C_m as follows:

$$C_m = f(S_m * F) \tag{20}$$

where F denotes the filter and f represents the RELU activation function. It replaces negative values with zeros reducing the non-linearity of the network. The detected n-grams achieving the highest score are then passed to the max pooling layer of dimension 2x2 and stride of 2 which reduces the size of the data.

Prediction based on Bi-LSTM Network. The output generated from the max pooling layer is concatenated and acts as input to this stage. Every word in the feature set is fed into the Bi-LSTM network cell. Bi-LSTM carries slightly different hidden computations. After that dropout (0.2) layer is implemented to prevent our model from overfitting. Every current input (or neuron) in the layer of the network is connected to every input (or neuron) in the layer that comes after it using the dense layer, also referred to as the completely connected layer. This network uses two dense layers. The final layer is the Softmax layer. It is often the last layer in any neural network. To achieve the desired output, the Softmax layer uses sigmoid as an activation function to assign classes to vulnerability prediction.

Smaller filter sizes, such as 2×2, were found to be inadequate in capturing sufficient contextual information, while larger filter sizes like 5×5 or 6×6 added unnecessary complexity without yielding significant improvements in accuracy. A 4×4 filter size offered the optimal balance, effectively capturing relevant word patterns while maintaining computational efficiency (Johnson & Zhang, 2014; Kalchbrenner et al., 2014; Zhang & Wallace, 2015). Furthermore, increasing the number of filters enhances the model's ability to

capture detailed features, but it also raises the risk of overfitting and increases computational demands. A configuration of 256 filters in the first layer, to capture a wide range of initial patterns, followed by 128 filters in the second layer to refine these patterns, was found to strike the best balance between performance and computational efficiency (Johnson & Zhang, 2014; Santos & Gatti, 2014; Zhang & Wallace, 2015).

The choice of a 2×2 max-pooling size with a stride of 2 is consistent with standard CNN architectures, as it effectively reduces the spatial dimensions of the feature maps while preserving important information (LeCun et al., 1998). A dropout rate of 0.2 was selected to mitigate overfitting and ensure better generalization during training (Hochreiter, 1997). This allows the model to capture essential features while maintaining computational efficiency and preventing overfitting, ultimately improving the overall performance of the vulnerability prediction task.

4.2 Main scenario 2: FWFS module

Filter methods for feature selection (FS) are computationally inexpensive and rapid. However, they may not always yield the optimal performance for SVSPMs. On the other hand, wrapper methods can achieve better classification performance for SVSPMs but are computationally intensive and time-consuming. To leverage the benefits of both approaches, an FWFS module is proposed, which integrates filter and wrapper methods to assess the fitness of feature subsets. The proposed architecture consists of three sections, illustrated in Fig. 5 and elaborated upon below:

1. Data generation: The first step is to generate input data from the raw unstructured data available. This requires a series of sub-steps. As we have unstructured data, there is a need to preprocess this data. The traditional preprocessing steps for unstructured data are tokenization, stop word removal, and stemming. After applying all these steps, we are left with a reduced set of tokens compared with the original data set. For vectorization unigram Bag-Of -Word (BoW) model is used which works on the vocabulary of features. For example, 'Sudo before 1.9.13 does not escape control characters in sudoreplay output.' is the vulnerability description for Vulnerability 1 (V1), and 'Sudo before 1.9.13 does not escape control characters in log messages.' is the vulnerability description for Vulnerability 2 (V2). Now V1 and V2 are represented as [1 1 1 1 1 1 0 0 1 1 1 1 1] and [1 1 1 1 1 1 1 1

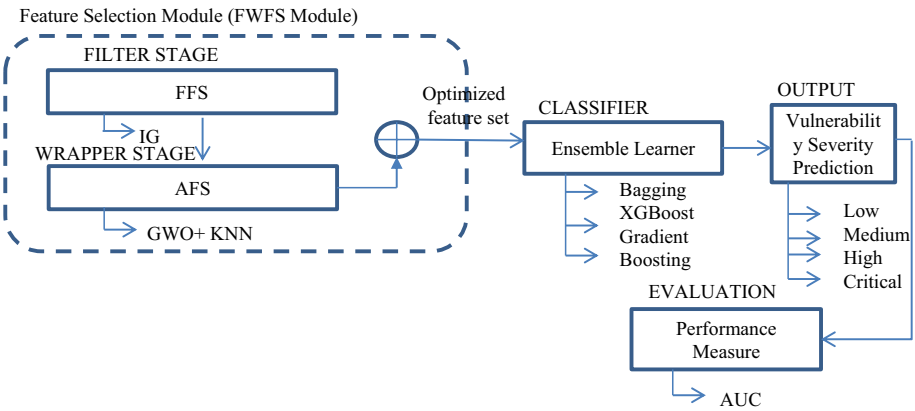

Fig. 5 Framework of FWFS module

1 0 1 0 1] respectively on the basis of vocabulary < before characters control does escape in log messages, not output sudo sudoreplay 1.9.13 > of size 13. These vectors are then weighted according to their frequency in the text using the TF-IDF approach, and feature vectors are formed. It is considered to be a powerful and easy statistical method for term weighting (Ramos, 2003; Witten et al., 2016).

2. Feature Selection: The second step focuses on reducing the number of features, known as dimensionality reduction. This term was first coined by Richard Bellman (1957). It is very important to reduce dimensionality, eliminating those words that possess no or least information from the text. The main concern of the filter part of the FWFS module is to fasten the process fitness evaluation. Therefore, the IG method, a computationally cheap method, is used to form the fitness evaluation based on the Min-Redundancy Max-Relevance (MRMR) criterion (Long & Ding, 2005) acting as an FFS. Feature vectors generated in Step 1 are used to evaluate the IG of all features using Eq. (4), and the features subset having the highest value for IG is finally selected. In the wrapper part of the FWFS module bGWO technique is used. It is a newly-proposed swarm intelligence technique analogous to the hunting and dominancy behavior of grey wolves in nature (Faris et al., 2018). This algorithm has been used to a vast number of issues in the literature. Important features are extracted through the GWO technique which acts as AFS with fitness function as minimizing the error rate of K-NN where K-NN is applied as the final classifier. As shown in Eq. (21), the following fitness function is utilized in this paper.

$$Fitness(X) = (\alpha * ErrorRate) + (1 - \alpha)\frac{N - S}{S} \tag{21}$$

where X represents the subset of features chosen by the GWO algorithm, ErrorRate represents the K-NN classifier's error rate on the training dataset, α represents a factor that balances the error rate and the size of the features chosen, N represents the total number of features in the dataset, and S represents the size of the features subset.

The selection of the most significant features (words) based on bGWO is listed below in the following steps:

1. The initial position of the wolves is generated, which is the total number of words.
2. The importance of every wolf is calculated based on a fitness function.
3. Wolfs are classified into four groups alpha, beta, delta, and omega.
4. The movement of wolves is seen toward the best wolf of the group, i.e., the alpha wolf.
5. Wolves of low worth are deleted, and the best wolfs representing nothing, but the features based on dominance and crowding distance are selected.

Every algorithm starts with the setting of parameters and initialization of the population, randomly. Table 2 outlines the parameter settings used to conduct this study. All the parameters are either chosen based on domain-specific information, as in the case of the parameters α, β, or by doing small trial simulations, or they are chosen based on what is commonly found in the literature, as in the case of the remaining parameters. Through prior studies, it is observed that the Grey Wolf Optimizer (GWO) tends to converge within a reasonable number of iterations, and setting it to 70 is sufficient to achieve convergence

Table 2 List of parameters used

Repetition of Runs	20
Number of Iterations	70
Search Agents	8
Dimension	Number of features
Search Domain	[0 1]
α	0.99
B	0.01

in many cases. Our primary goal is to compare the performance of the various feature selection problems compared to the proposed module for feature selection.

FS selects the features that boost and maximize the performance of the classifier. Binary chromosome illustration is utilized to depict the feature subset selection as depicted in Fig. 6. The chromosome's length indicates the total number of features. The value '1' suggests selecting the feature and the value '0' suggests rejecting the feature.

3. As shown in Table 1, the dataset exhibits imbalance. To address this imbalance, SMOTE, an oversampling strategy, is employed.

4. Classification: When the input data is reduced and balanced, it is fed into the classifier, and the final output is generated. Instead of using simple ML algorithms to develop a classifier, more powerful ensemble learners (Schirrmeister et al., 2017) are exploited. Among the ensemble learning techniques investigated, we have delved into Bagging and Boosting methods, including Gradient Boosting Decision Trees (GBDT) and XGBoost due to their exceptional predictive performance, robustness to overfitting, flexibility in handling diverse data types, feature importance analysis capabilities, and widespread adoption in both academia and industry. The outcomes are documented using Area Under the Curve (AUC) as a metric for performance evaluation. Following the recording of results for the primary scenarios, an additional five scenarios are executed, with subsequent recording and analysis of the results. For hypothesis testing, statistical analysis is performed to judge whether the performance of different strategies differs significantly or not. For this purpose, we need to frame a null hypothesis (Ho) and an alternate hypothesis (H1). Both are stated below.

Ho: The performance of SVSPMs developed using different FS strategies is the same and does not differ significantly.

RQ1. How does SVSPM perform better when different FS methods are used to make the data less dimensional?

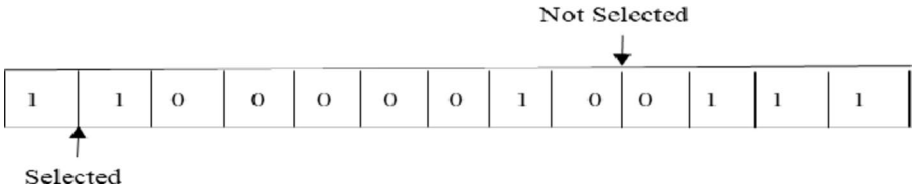

Fig. 6 Solution representation

Table 3 AUC values for Mozilla Firefox

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.86	0.84	0.7	0.91	0.83
	CNN	0.85	0.81	0.7	0.92	0.82
HDLN	CNN + Bi-LSTM	0.89	**0.86**	0.76	0.92	0.86
Filter Method	IG + Bagging	0.9	0.73	0.89	0.92	0.86
	IG + GB	0.94	0.77	0.89	0.91	0.88
	IG + XGB	0.93	0.77	0.88	0.92	0.88
Wrapper Method	GWO + Bagging	0.89	0.72	0.85	0.88	0.84
	GWO + GB	0.94	0.78	0.88	0.91	0.88
	GWO + XGB	0.95	0.76	0.89	0.92	0.88
FWFS	GWO + IG + Bagging	0.91	0.75	0.85	0.89	0.85
	GWO + IG + GB	0.94	0.76	0.87	0.92	0.87
	GWO + IG + XGB	**0.95**	0.78	**0.9**	**0.93**	**0.89**
No Feature Selection	Bagging	0.79	0.72	0.79	0.86	0.79
	GB	0.76	0.7	0.77	0.87	0.78
	XGB	0.79	0.72	0.82	0.89	0.81

Table 4 AUC values for Mozilla Thunderbird

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.92	0.78	0.83	0.95	0.87
	CNN	0.89	0.73	0.76	0.94	0.83
HDLN	CNN + Bi-LSTM	**0.95**	**0.81**	0.86	**0.95**	**0.89**
Filter Method	IG + Bagging	0.78	0.61	0.81	0.83	0.76
	IG + GB	0.86	0.68	**0.93**	0.87	0.84
	IG + XGB	0.8	0.66	0.9	0.86	0.81
Wrapper Method	GWO + Bagging	0.74	0.63	0.87	0.8	0.76
	GWO + GB	0.83	0.68	0.93	0.87	0.83
	GWO + XGB	0.83	0.66	0.9	0.87	0.82
FWFS	GWO + IG + Bagging	0.83	0.68	0.89	0.82	0.81
	GWO + IG + GB	0.81	0.69	0.88	0.85	0.81
	GWO + IG + XGB	0.85	0.7	0.9	0.88	0.83
No Feature Selection	Bagging	0.59	0.63	0.68	0.78	0.67
	GB	0.41	0.66	0.79	0.79	0.66
	XGB	0.5	0.66	0.78	0.8	0.69

The result values are recorded in Tables 3, 4, 5, 6, 7 for AUC and Tables 8, 9, 10, 11, 12 for the F1 measure. When AUC values are observed for the Mozilla Firefox vulnerability database, the highest values obtained are 0.95, 0.86, 0.90, 0.93, and 0.89 for GWO + IG + GXB, CNN + Bi-LSTM, GWO + IG + GXB, GWO + IG + GXB, GWO + IG + GXB models respectively. 0.76, 0.70, 0.70,0.86, and 0.78 are the lowest values obtained using NOFS + GB, NOFS + GB, Bi-LSTM, and CNN, NOFS + Bagging NOFS + GB. NOFS stands for No FS. The result values are in the sequence of a low,

Table 5 AUC values for Mozilla Seamonkey

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.86	0.81	0.72	0.92	0.83
	CNN	0.86	0.72	0.71	**0.94**	0.81
HDLN	CNN + Bi-LSTM	**0.95**	**0.82**	**0.86**	**0.94**	**0.89**
Filter Method	IG + Bagging	0.66	0.65	0.79	0.72	0.71
	IG + GB	0.78	0.71	0.82	0.81	0.78
	IG + XGB	0.73	0.68	0.8	0.8	0.75
Wrapper Method	GWO + Bagging	0.64	0.66	0.75	0.74	0.70
	GWO + GB	0.74	0.7	0.83	0.82	0.77
	GWO + XGB	0.8	0.68	0.83	0.8	0.78
FWFS	GWO + IG + Bagging	0.68	0.65	0.78	0.72	0.71
	GWO + IG + GB	0.81	0.72	0.82	0.82	0.79
	GWO + IG + XGB	0.82	0.73	0.85	0.84	0.81
No Feature Selection	Bagging	0.46	0.61	0.54	0.71	0.58
	GB	0.73	0.6	0.71	0.72	0.69
	XGB	0.63	0.61	0.74	0.71	0.67

Table 6 AUC values for Mozilla Firefox ESR

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.96	0.78	0.87	0.95	0.89
	CNN	0.95	**0.8**	0.81	0.9	0.87
HDLN	CNN + Bi-LSTM	**0.97**	**0.8**	**0.91**	**0.97**	**0.91**
Filter Method	IG + Bagging	0.76	0.71	0.78	0.86	0.78
	IG + GB	0.8	0.69	0.81	0.92	0.81
	IG + XGB	0.75	0.66	0.81	0.9	0.78
Wrapper Method	GWO + Bagging	0.75	0.68	0.75	0.9	0.77
	GWO + GB	0.76	0.68	0.8	0.92	0.79
	GWO + XGB	0.78	0.65	0.83	0.9	0.79
FWFS	GWO + IG + Bagging	0.73	0.73	0.84	0.83	0.78
	GWO + IG + GB	0.75	0.69	0.82	0.85	0.78
	GWO + IG + XGB	0.82	0.68	0.83	0.94	0.82
No Feature Selection	Bagging	0.71	0.66	0.75	0.9	0.76
	GB	0.72	0.65	0.79	0.9	0.77
	XGB	0.73	0.68	0.8	0.87	0.77

medium, high, and critical severity level. The fifth value is the average result of all these four severity levels. A similar analysis for the remaining four databases is documented further in the text. For Mozilla Thunderbird, 0.95, 0.81, 0.93, 0.95, 0.89 are the highest values obtained with the use of CNN + Bi-LSTM, CNN + Bi-LSTM, IG + GB, CNN + Bi-LSTM, and CNN + Bi-LSTM respectively. 0.41, 0.61, 0.68, 0.78, and 0.66 are the lowest values obtained with the use of NOFS + GB, IG + Bagging, NOFS + Bagging, NOFS + Bagging, and NOFS + GB respectively. For Mozilla Seamonkey, 0.97, 0.80, 0.91, 0.97, and 0.91 are

Table 7 AUC values for Mozilla Thunderbird ESR

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	Nan	0.83	0.76	0.85	0.81
	CNN	Nan	0.9	0.89	0.94	0.91
HDLN	CNN + Bi-LSTM	Nan	**0.9**	0.9	**0.95**	**0.92**
Filter Method	IG + Bagging	Nan	0.66	0.8	0.77	0.74
	IG + GB	Nan	0.72	0.98	0.8	0.83
	IG + XGB	Nan	0.76	**0.97**	0.83	0.85
Wrapper Method	GWO + Bagging	Nan	0.65	0.82	0.74	0.74
	GWO + GB	Nan	0.68	**0.97**	0.78	0.81
	GWO + XGB	Nan	0.77	**0.97**	0.83	0.86
FWFS	GWO + IG + Bagging	Nan	0.64	0.82	0.73	0.73
	GWO + IG + GB	Nan	0.72	**0.97**	0.8	0.83
	GWO + IG + XGB	Nan	0.76	**0.97**	0.83	0.85
No Feature Selection	Bagging	Nan	0.61	0.62	0.7	0.64
	GB	Nan	0.55	0.56	0.59	0.57
	XGB	Nan	0.62	0.79	0.69	0.70

Table 8 F1 measure for Mozilla Firefox

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.8	0.79	0.66	0.88	0.78
	CNN	0.81	0.77	0.65	0.86	0.77
HDLN	CNN + Bi-LSTM	0.8	**0.82**	0.72	**0.89**	0.81
Filter Method	IG + Bagging	0.85	0.69	0.83	0.88	0.81
	IG + GB	0.9	0.74	0.85	0.87	0.84
	IG + XGB	0.88	0.72	0.8	0.85	0.81
Wrapper Method	GWO + Bagging	0.84	0.68	0.82	0.81	0.79
	GWO + GB	0.78	0.71	0.79	0.8	0.77
	GWO + XGB	0.85	0,68	0.77	0.83	0.82
FWFS	GWO + IG + Bagging	0.87	0.72	0.79	0.85	0.81
	GWO + IG + GB	0.9	0.71	0.83	0.88	0.83
	GWO + IG + XGB	**0.89**	0.75	**0.85**	**0.89**	**0.85**
No Feature Selection	Bagging	0.77	0.66	0.76	0.83	0.76
	GB	0.7	0.65	0.74	0.82	0.73
	XGB	0.7	0.7	0.77	0.83	0.75

the highest values obtained using the CNN + Bi-LSTM model for all four severity levels and the average case. 0.46, 0.60, 0.54, 0.71, and 0.58 are the lowest values obtained with the use of NOFS + Bagging, NOFS + GB, NOFS + Bagging, NOFS + GB, and NOFS + Bagging respectively. For Mozilla Firefox ESR, 0.97, 0.80, 0.91, 0.97, and 0.91 are the highest values obtained using the CNN + Bi-LSTM model for all four severity levels and as well as for average cases. 0.71, 0.65, 0.75, 0.83, and 0.76 are the lowest values obtained with the use of NOFS + Bagging, NOFS + GB, NOFS + Bagging, GWO + IG + Bagging, and

Table 9 F1 measure for Mozilla Thunderbird

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.85	0.75	0.8	0.8	0.80
	CNN	0.86	0.68	0.71	0.88	0.78
HDLN	CNN + Bi-LSTM	**0.9**	**0.76**	0.8	**0.89**	**0.84**
Filter Method	IG + Bagging	0.71	0.6	0.77	0.79	0.72
	IG + GB	0.85	0.66	**0.89**	0.81	0.80
	IG + XGB	0.77	0.65	0.85	0.83	0.78
Wrapper Method	GWO + Bagging	0.7	0.6	0.82	0.79	0.73
	GWO + GB	0.78	0.62	0.87	0.83	0.78
	GWO + XGB	0.78	0.64	0.84	0.81	0.77
FWFS	GWO + IG + Bagging	0.79	0.64	0.83	0.77	0.76
	GWO + IG + GB	0.78	0.64	0.83	0.81	0.77
	GWO + IG + XGB	0.81	0.67	0.86	0.82	0.79
No Feature Selection	Bagging	0.55	0.6	0.61	0.77	0.63
	GB	0.4	0.62	0.77	0.74	0.63
	XGB	0.5	0.62	0.75	0.78	0.66

Table 10 F1 measure for Mozilla Seamonkey

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.81	0.76	0.69	**0.88**	0.79
	CNN	0.8	0.7	0.67	**0.88**	0.76
HDLN	CNN + Bi-LSTM	**0.91**	**0.78**	**0.82**	**0.88**	**0.85**
Filter Method	IG + Bagging	0.65	0.65	0.75	0.7	0.69
	IG + GB	0.73	0.66	0.78	0.79	0.74
	IG + XGB	0.68	0.66	0.78	0.77	0.72
Wrapper Method	GWO + Bagging	0.61	0.62	0.71	0.69	0.66
	GWO + GB	0.7	0.69	0.78	0.77	0.74
	GWO + XGB	0.78	0.65	0.78	0.79	0.75
FWFS	GWO + IG + Bagging	0.65	0.61	0.73	0.69	0.67
	GWO + IG + GB	0.78	0.68	0.78	0.77	0.75
	GWO + IG + XGB	0.77	0.69	0.81	0.79	0.77
No Feature Selection	Bagging	0.44	0.6	0.5	0.68	0.56
	GB	0.69	0.55	0.65	0.69	0.65
	XGB	0.6	0.58	0.71	0.67	0.64

NOFS + Bagging respectively. For Mozilla Thunderbird ESR, 0.90, 0.97, 0.95, and 0.92 are the highest values obtained using CNN + Bi-LSTM, CNN + Bi-LSTM, and CNN + Bi-LSTM for the medium and critical levels of severity. The third value is for the average case. For the high severity level, 0.97 is the highest value obtained using several models, which are IG + XGB, GWO + GB, GWO + XGB, GWO + IG + GB, and GWO + IG + XGB. 0.55, 0.56, 0.59, and 0.57 are the lowest values obtained with the use of the NOFS + GB model for all levels and as well as for the average case. For this database, we could not get

Table 11 F1 measure for Mozilla Firefox ESR

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	0.91	0.72	0.83	0.89	0.84
	CNN	0.88	0.75	0.77	0.81	0.80
HDLN	CNN + Bi-LSTM	**0.93**	**0.76**	**0.86**	**0.93**	**0.87**
Filter Method	IG + Bagging	0.71	0.66	0.73	0.83	0.73
	IG + GB	0.77	0.65	0.78	0.86	0.77
	IG + XGB	0.71	0.62	0.77	0.85	0.74
Wrapper Method	GWO + Bagging	0.71	0.63	0.71	0.85	0.73
	GWO + GB	0.73	0.62	0.76	0.87	0.75
	GWO + XGB	0.75	0.61	0.77	0.86	0.75
FWFS	GWO + IG + Bagging	0.7	0.7	0.81	0.8	0.75
	GWO + IG + GB	0.72	0.63	0.78	0.81	0.74
	GWO + IG + XGB	0.77	0.65	0.8	0.9	0.78
No Feature Selection	Bagging	0.68	0.66	0.7	0.85	0.72
	GB	0.68	0.61	0.73	0.87	0.72
	XGB	0.69	0.65	0.77	0.83	0.74

Table 12 F1 measure for Mozilla Thunderbird ESR

Feature Selection	Model	Low	Medium	High	Critical	AVERAGE
Deep Learning	Bi-LSTM	Nan	0.77	0.7	0.8	0.76
	CNN	Nan	0.81	0.8	0.79	0.80
HDLN	CNN + Bi-LSTM	Nan	**0.88**	0.84	**0.89**	**0.87**
Filter Method	IG + Bagging	Nan	0.62	0.77	0.72	0.70
	IG + GB	Nan	0.68	0.92	0.76	0.79
	IG + XGB	Nan	0.7	0.92	0.79	0.80
Wrapper Method	GWO + Bagging	Nan	0.6	0.74	0.7	0.68
	GWO + GB	Nan	0.63	0.91	0.72	0.75
	GWO + XGB	Nan	0.73	**0.93**	0.77	0.81
FWFS	GWO + IG + Bagging	Nan	0.6	0.8	0.68	0.69
	GWO + IG + GB	Nan	0.66	**0.93**	0.77	0.79
	GWO + IG + XGB	Nan	0.7	**0.93**	0.75	0.79
No Feature Selection	Bagging	Nan	0.56	0.6	0.66	0.61
	GB	Nan	0.51	0.5	0.55	0.52
	XGB	Nan	0.58	0.73	0.62	0.64

results for a low level of severity as it had very few numbers of instances under it. More or less similar patterns were seen for the values of the F1 measure.

In a nutshell, out of the total 19 best values, 14 values are given by the model using the HDLN module for FS, four values are given by the model using the FWFS module for FS, and the remaining one value which is for high severity level of Mozilla Thunderbird database is given by IG + GB. The value is 0.93. However, the second-highest value is 0.90, given by the FWFS module. If we look at average case values for all five databases,

referring to Table 13, four values are obtained by the HDLN module, and one value is obtained by the FWFS module. Similarly, if we briefly analyze the lowest values based on average case values, all five values are obtained when no FS method is applied. Therefore, referring to the above result analysis, it can be stated that our proposed models definitely helped in improving the efficiency of SVSPM. A deeper analysis of results is done in further RQs below.

RQ1.1 *When the HDLN module is used to lower the dimensionality of the data, what performance improvement does SVSPM experience?*

To address this RQ, Fig. 7. is constructed using values from Table 13. It compares the performance of the model constructed using the HDLN module with the CNN model and Bi-LSTM model. It is clearly depicted in Fig. 7. that the model using the HDLN module gave the best performance of 0.86, 0.88, 0.88, and 0.88 for Mozilla Firefox, Mozilla Thunderbird, Mozilla Seamonkey, Mozilla Firefox ESR, and Mozilla Thunderbird ESR respectively. From the Bi-LSTM model, the HDLN module gave a percentage improvement of 3.61%, 2.24%, 3.40%, 3.40%, and 2.23% for all five databases respectively. From the CNN model, the HDLN module gave a percentage improvement of 4.65%, 6.74%, 5.68%, 5.68%, and 5.68% for all five databases respectively.

This analysis demonstrates the superiority of the HDLN module in terms of overall classification performance. The module is more effective in capturing features and making accurate predictions across different Mozilla databases, particularly when compared to traditional models like CNN and Bi-LSTM. Thus, the HDLN module proves to be a highly efficient and robust solution for the task at hand.

RQ1.2 When the FWFS module is used to lower the dimensionality of the data, what performance improvement does SVSPM experience?

When Fig. 8. is analyzed it can be observed that, for all five databases, models developed without FS, irrespective of classifier used gave a poor performance (depicted by blue line in the graphs) and models developed using FWFS module gave the best performance (depicted by yellow line in the graph).

Table 13 Average result values

Feature Selection	Model	MF	MT	MS	MF ESR	MT ESR	AVERAGE
Deep Learning	Bi-LSTM	0.83	0.87	0.85	0.85	0.86	0.85
	CNN	0.82	0.83	0.83	0.83	0.83	0.83
HDLN	CNN + Bi-LSTM	0.86	**0.89**	**0.88**	**0.88**	**0.88**	**0.88**
Filter Method	IG + Bagging	0.86	0.76	0.81	0.81	0.79	0.81
	IG + GB	0.88	0.84	0.86	0.86	0.85	0.85
	IG + XGB	0.88	0.81	0.84	0.84	0.83	0.84
Wrapper Method	GWO + Bagging	0.84	0.76	0.80	0.80	0.79	0.80
	GWO + GB	0.88	0.83	0.85	0.85	0.84	0.85
	GWO + XGB	0.88	0.82	0.85	0.85	0.84	0.85
FWFS	GWO + IG + Bagging	0.85	0.81	0.83	0.83	0.82	0.83
	GWO + IG + GB	0.87	0.81	0.84	0.84	0.83	0.84
	GWO + IG + XGB	**0.89**	0.83	0.86	0.86	0.85	0.86
No Feature Selection	Bagging	0.79	0.67	0.73	0.73	0.71	0.73
	GB	0.78	0.66	0.72	0.72	0.70	0.72
	XGB	0.81	0.69	0.75	0.75	0.73	0.74

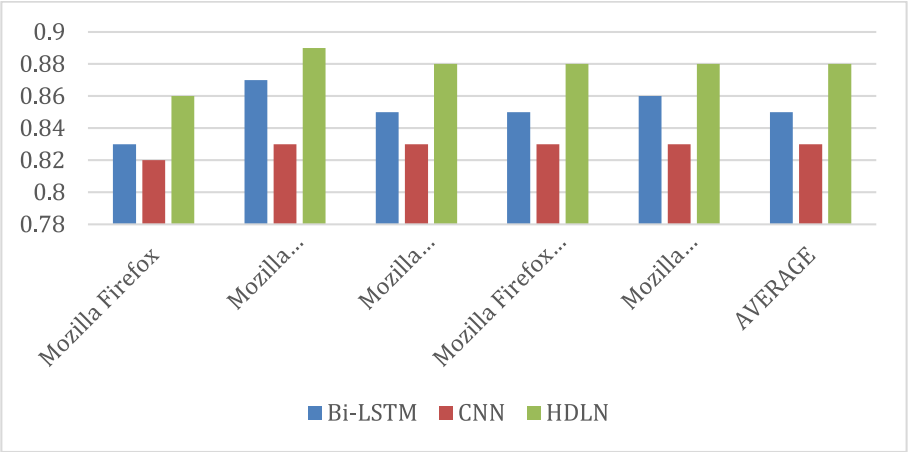

Fig. 7 AUC values comparing performance of HDLN module with CNN & BI-LSTM model

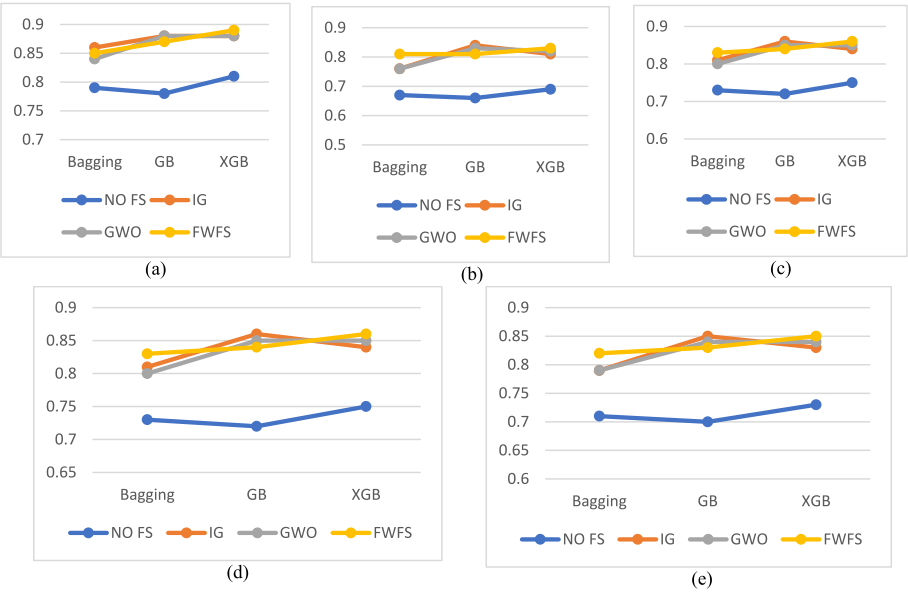

Fig. 8 AUC values comparing performance of FWFS module with No FS, IG, and GWO methods of FS for (**a**) Mozilla Firefox (**b**) Mozilla Thunderbird (**c**) Mozilla Seamonkey (**d**) Mozilla Firefox ESR (**e**) Mozilla Thunderbird ESR

The average percentage improvement of 9.58%, 18.06% and 14.86% is recorded when IG FS method is applied over bagging, GB, and XGB model. The average percentage improvement of 10.95%, 18.06% and 13.51% is recorded when GWO FS method is applied over bagging, GB, and XGB model. The average percentage improvement of 13.70%, 16.67% and 16.22% is recorded when FWFS module is applied over bagging, GB, and XGB model.

In conclusion, feature selection is crucial for enhancing model performance, and the FWFS module consistently delivers the best results across all classifiers. The FWFS module outperformed both the IG and GWO methods, demonstrating its superior ability to identify relevant features and improve classification accuracy. These findings highlight the importance of incorporating advanced FS techniques to achieve optimal model performance.

RQ2. Which module of FS among the two proposed modules gave the best results?

To address RQ2, graph of Fig. 9. is constructed based on percentage improvement in the performance of the model using HDLN module and FWFS module for FS from the model using no FS. We have one model under the HDLN module, i.e., the CNN + Bi-LSTM model, and three models under the FWFS module, i.e., GWO + IG + Bagging, GWO + IG + GB, GWO + IG + XGB. For Mozilla Firefox, GWO + IG + XGB has the highest improvement of 9.88, followed by GWO + IG + GB with 7.72.

For the other four databases, highest percentage improvement is given by CNN + Bi-LSTM model. The values are 29.35, 29.35, 18.51, and 30.95 for Mozilla Sea monkey, Mozilla Thunderbird, Mozilla Firefox ESR, and Mozilla Thunderbird ESR. GWO + IG + GB gave the second-highest improvement for the remaining four databeses. The values are 20.65, 17.95, 6.17, and 21.98 for Mozilla Sea monkey, Mozilla Thunderbird, Mozilla Firefox ESR, and Mozilla Thunderbird ESR, respectively.

Thus, it can be summarized and stated that out of five cases, the model belonging to the HDLN module gave the best results, hence HDLN module has better performance than the FWFS module. If the average result values of all five databases are further averaged, then also HDLN gave the highest result of 0.88, and the IG + GWO + XGB model of the FWFS module gave the second-highest result of 0.86. These values are recorded in Table 13.

On the other hand, this analysis highlights that, although the CNN + Bi-LSTM model offers the most robust performance improvement overall, the FWFS module, particularly when using the GWO + IG + XGB and GWO + IG + GB combinations, also provides substantial enhancements, making it a competitive alternative.

RQ3. Whether the performance of SVSPM developed boosted significantly with the help of proposed FS modules used to reduce the dimensionality of data?

To address RQ3, null and alternate hypotheses are formulated as follows:

H0: The performance of SVSPMs developed using different FS strategies is the same and does not differ significantly.

H1: The performance of SVSPMs developed using different FS strategies differs significantly.

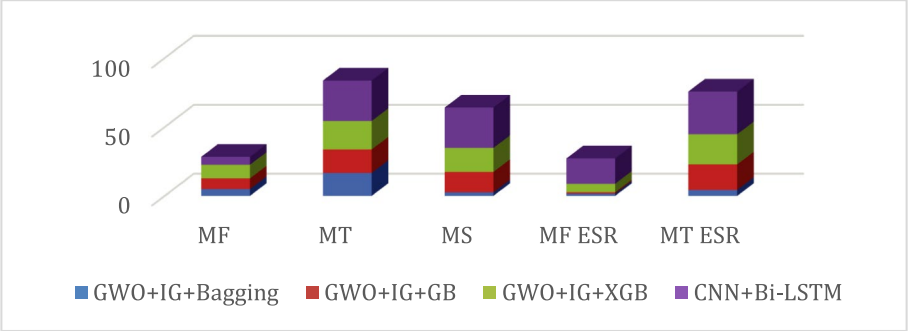

Fig. 9 Percentage improvement of HDLN module and FWFS module with model using No FS

To test the above-stated hypothesis, α is considered to be 0.05, i.e., 95% confidence level. In Table 14, the results of the Friedman test are documented. The mean rank for all four levels of severity and the average case for all models are documented databases-wise. A large value of rank obtained by a particular model indicates that that model is better.

When Friedman's test was conducted, considering all scenarios of FS on five different databases used in this study, the p-value came out to be 0.00($p < 0.05$). The p-value indicates that the results of Friedman's test are significant. It can be seen from Friedman's test result table that the best rank is achieved by the CNN-BiLSTM model of 12.88, 14.80, 14.10, and 13.60 for low, medium, critical level, and average cases. For a high level, the best rank of 12.90 was achieved by the GWO + IG + XGB model. Worst ranks are obtained for no FS.

As Friedman's test results are significant, the null hypothesis (H0) is rejected, and the alternate hypothesis (H1) is accepted. Hence, it can be stated that significant improvement is observed in the performance of SVSPM developed after applying the proposed modules for FS. HDLN module emerges as the best module to improve the performance of SVSPM, followed by GWO + IG + XGB, which comes under the FWFS module.

5 Threats to validity

Threats to the validity of our study are discussed in this section. Empirical software engineering, according to Wohlin (2007), (Johnson & Zhang, 2014), is essential to any empirical study to identify and address threats to the validity of the experimental findings. The following are some potential threats to the validity of the results.

Software vulnerabilities of five different products of Mozilla are collected, and experiments are performed on those. There is a possibility that new vulnerabilities may be reported now and are not present in the database collected by us, leading to a threat to validity. The database contains vulnerabilities in five different products of Mozilla. Therefore, the results

Table 14 Friedman's test result

Model	Mean Rank (Low)	Mean Rank (Medium)	Mean Rank (High)	Mean Rank (Critical)	Mean Rank (Average)
Bi-LSTM	11.63	13.60	5.70	12.30	11.00
CNN	10.88	13.30	4.40	12.10	10.80
CNN + Bi-LSTM	12.88	14.80	9.80	14.10	13.60
IG + Bagging	6.38	5.50	6.80	6.10	5.90
IG + GB	11.00	9.30	12.20	9.20	11.30
IG + XGB	7.13	7.30	10.30	9.10	8.60
GWO + Bagging	4.88	4.60	5.90	5.00	4.70
GWO + GB	9.38	8.60	11.00	9.20	9.40
GWO + XGB	10.88	6.80	12.20	9.50	10.90
GWO + IG + Bagging	6.63	7.10	8.80	3.70	6.30
GWO + IG + GB	9.00	9.60	10.10	7.90	8.70
GWO + IG + XGB	12.38	10.70	12.90	12.60	12.70
Bagging	1.88	2.70	2.10	2.80	1.60
GB	2.63	2.00	3.10	3.30	1.60
XGB	2.50	4.10	4.70	3.10	2.90

of this study may be generalized to the domains of Mozilla products but may differ for other domains posing a threat to external validity. The use of open-source and publicly available datasets reduces the threat to construct validity. To generalize results, the experiment is carried out by using an FS module that works in two stages, trying to handle the curse of dimensionality problem. Vulnerability description is used among so many fields available in vulnerability reports to carry out the experiment posing a threat to the validity of results. To reduce the risk of validity of the conclusion, tenfold cross-validation statistical tests are applied during the development of SVSPM. AUC performance measure is used to evaluate the effectiveness of the SVSPM developed. AUC finds a balance between both specificity and sensitivity values, reducing the threat to the validity of results.

6 Concluding remarks

As software complexity rises, achieving error-free software becomes nearly unattainable. Consequently, software testing persists even after release to uncover potential vulnerabilities. The prevalence of vulnerability data in textual format presents a challenge for prediction models due to dimensionality issues. To tackle this challenge, the study introduces two Feature Selection (FS) modules designed to alleviate dimensionality concerns and enhance the performance of prediction models. The proposed FS modules, HDLN and FWFS, employ distinct approaches: HDLN utilizes the CNN-Bi-LSTM model for SVSPM development, while FWFS employs IG + GWO for FS. To assess the efficiency of these modules' models, various alternative models are developed, including those utilizing CNN only, Bi-LSTM only, IG only, GWO only, and ensemble classifiers like Bagging, GB, and XGB. Due to dataset imbalance, SMOTE is applied as a data balancing technique. The dataset is then divided into training and testing sets using tenfold cross-validation. Evaluating the findings of SVSPM across four severity levels involves recording corresponding AUC values. Statistical tests are employed to bolster the reliability of the results. The analysis reveals that the application of FS methods significantly enhances SVSPM performance compared to models developed without FS. Among these, the HDLN module yields the most favorable results, followed by the FWFS module.

The proposed approach has significant practical applications in cybersecurity, software maintenance, and risk management. The primary goal of the developed model is to automate the process of assigning priority levels to vulnerabilities based on descriptions found in databases like CVE (Common Vulnerabilities and Exposures) or bug reports. Instead of relying on manual analysis, the model employs machine learning to extract key information from text and determine the urgency or severity of each vulnerability. This enables faster, more efficient security management. By using historical data from past vulnerabilities, the model improves the accuracy of prioritization, predicting the criticality of new vulnerabilities. Its ability to streamline feature selection enhances the accuracy and efficiency of vulnerability severity predictions, which is crucial for prioritizing security patches and mitigating high-risk vulnerabilities in large software systems. In cybersecurity, this can improve automated vulnerability management and proactive defense strategies.

This approach helps organizations allocate patching resources more effectively by focusing on high-risk vulnerabilities that could significantly impact their systems. As a result, critical vulnerabilities can be addressed first, ensuring security teams concentrate on the most pressing issues. The demonstrated improvement in predictive performance (AUC values of 0.88 and 0.86) underscores the model's practical potential in real-world scenarios.

Our research addresses the obstacles posed by high-dimensional textual data through the introduction of an effective Feature Selection (FS) module. Our future endeavors entail investigating multiple data balancing techniques, including SMOTE, ADASYN, ROS, RUS, SMOTEBoost, and EasyEnsemble, to further improve our SVSPM. Additionally, we aim to develop a specialized data balancing module tailored to meet the specific requirements of SVSPM.

Authors' contribution Ruchika Malhotra and Vidushi conceptualized the idea presented in the manuscript. Vidushi carried out the experiments and wrote the first draft of the manuscript. Ruchika Malhotra supervised the work, reviewed the draft and finalized the manuscript. Both authors read and approved the final manuscript.

Funding The author did not receive support from any organization for the submitted work.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of Interest The authors declare no competing interests.

References

Alomari, O. A., Elnagar, A., Afyouni, I., Shahin, I., Nassif, A. B., Hashem, I. A., & Tubishat, M. (2022). Hybrid feature selection based on principal component analysis and grey wolf optimizer algorithm for Arabic news article classification. *IEEE Access, 10*, 121816–121830.

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. https://doi.org/10.1109/EDCC.2016.34

Anju, A. J., & Judith, J. E. (2024). Hybrid feature selection method for predicting software defect. *Journal of Engineering and Applied Science, 71*(1), 124.

Ayetiran, E. F. (2022). Attention-based aspect sentiment classification using enhanced learning through CNN-BiLSTM networks. *Knowledge-Based Systems, 252*, 109409.

Bahad, P., Saxena, P. (2020). Study of AdaBoost and Gradient Boosting Algorithms for Predictive Analytics. In: International Conference on Intelligent Computing and Smart Communication. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-0633-8_22

Bellman, R. (1957). Dynamic programming. Princeton University press.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123–140. https://doi.org/10.1007/BF00058655

Chen, J., Kudjo, P. K., Mensah, S., Brown, S. A., & Akorfu, G. (2020). An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection. *Journal of Systems and Software, 167*, 110616.

Das, S. S., Serra, E., Halappanavar. M., Pothen, A., Al-Shaer, E., (2021). V2W-BERT: A framework for effective hierarchical multiclass classification of software vulnerabilities. In *IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–12. https://doi.org/10.1109/DSAA53316.2021.9564227

Davari, M., Zulkernine, M., Jaafar, F. (2017). An Automatic Software Vulnerability Classification Framework. In: International Conference on Software Security and Assurance (ICSSA), pp. 44–49. https://doi.org/10.1109/ICSSA.2017.27.

Du, J., Gui, L., Xu, R., and He. Y. (2017). A convolutional attention model for text classification. In: Proceedings of the National CCF Conference on Natural Language Processing and Chinese Computing. 183–195. https://doi.org/10.1007/978-3-319-73618–1_

Emary, E., Zawbaa, H.M., Grosan, C., Hassenian, A.E. (2015). Feature subset selection approach by grey-wolf optimization. In: Afro-European Conference for Industrial Advancement, pp. 1–13. https://doi.org/10.1007/978-3-319-13572-4_1

Erturk, E., & Akcapinar, E. (2015). A comparison of some soft computing methods for software fault prediction. *Expert syStems with Applications, 42*(4), 1872–1879. https://doi.org/10.1016/j.eswa.2014.10.025

Shahid, M.R., & Debar, H., (2021). "CVSS-BERT: Explainable Natural Language Processing to Determine the Severity of a Computer Security Vulnerability from its Description," *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Pasadena, CA, USA, pp. 1600–1607, https://doi.org/10.1109/ICMLA52953.2021.00256.

Faris, H., Aljarah, I., Al-Betar, M. A., & Mirjalili, S. (2018). Grey wolf optimizer: A review of recent variants and applications. *Neural Computing and Applications, 30*(2018), 413–435. https://doi.org/10.1007/s00521-017-3272-5

Firpi, H. A., & Goodman, E. (2004). Swarmed feature selection. In Proceedings of the 33rd applied imagery pattern recognition workshop, Washington, DC, USA. *IEEE Computer Society*, 112–118. https://doi.org/10.1109/AIPR.2004.41

Ghourabi, A., Mahmood, M. A., & Alzubi, Q. M. (2020). A hybrid CNN-LSTM model for SMS spam detection in Arabic and English messages. *Future Internet, 12*(9), 156. https://doi.org/10.3390/fi12090156

Hancer, E. (2024). An improved evolutionary wrapper-filter feature selection approach with a new initialisation scheme. *Machine Learning, 113*(8), 4977–5000.

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering, 21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

Hochreiter, S. (1997). *Long Short-term Memory*. Neural Computation MIT-Press.

Jain, D., Kumar, A., & Garg, G. (2020). Sarcasm detection in mash-up language using soft-attention based bi-directional LSTM and feature-rich CNN. *Applied Soft Computing, 91*, 106198.

Jeon, S., & Kim, H. K. (2021). AutoVAS: An automated vulnerability analysis system with a deep learning approach. *Computers & Security, 106*, 102308. https://doi.org/10.1016/j.cose.2021.102308

Johnson, R. and Zhang, T., (2014). Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint* arXiv:1412.1058.

Kalchbrenner, N., Grefenstette, E. and Blunsom, P., (2014). A convolutional neural network for modelling sentences. *arXiv preprint* arXiv:1404.2188.

Kekül, H., Ergen, B., & Arslan, H. (2021). A multiclass hybrid approach to estimating software vulnerability vectors and severity score. *Journal of Information Security and Applications, 63*, 103028.

Khazaei, A., Ghasemzadeh, M., & Derhami, V. (2016). An automatic method for CVSS scores prediction using vulnerabilities description. *Journal of Intelligent & Fuzzy Systems, 30*(1), 89–96. https://doi.org/10.3233/IFS-151733

Kim, H., & Jeong, Y. S. (2019). Sentiment classification using convolutional neural networks. *Applied Sciences, 9*(11), 2347.

Kudjo, P. K., Chen, J., Mensah, S., Amankwah, R., & Kudjo, C. (2020a). The effect of Bellwether analysis on software vulnerability severity prediction models. *Software Quality Journal, 28*, 1413–1446.

Kudjo, P. K., Aformaley, S. B., Mensah S., Chen, J. (2019). The Significant Effect of Parameter Tuning on Software Vulnerability Prediction Models. In: IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 526–527. https://doi.org/10.1109/QRS-C.2019.00107.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems, 33*(12), 6999–7019.

Liu, H., Motoda, H., Setiono, R., Zhao, Z. (2010). Feature selection: an ever-evolving frontier in data mining. In *Proc. the fourth workshop on feature selection in data mining*, vol. 4, pp. 4–13.

Liu, H., & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *Knowledge and Data Engineering, IEEE Transactions on, 17*(4), 491–502. https://doi.org/10.1109/TKDE.2005.66

Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and minredundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(8), 1226–1238. https://doi.org/10.1109/TPAMI.2005.159

Malhotra R., Vidushi (2021). Severity Prediction of Software Vulnerabilities Using Textual Data. In: International Conference on Recent Trends in Machine Learning, IoT, Smart Cities, and Applications. Advances in Intelligent Systems Computing, vol 1245. Springer, Singapore, https://doi.org/10.1007/978-981-15-7234-0_41

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer. *Advances in Engineering Software., 69*, 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

Olson, D. L., & Delen, D. (2008). Performance evaluation for predictive modelling. *Advanced Data Mining Techniques,* pp. 137–147.

Ramos, J. (2003). Using TF-IDF to determine word relevance in document queries. *Proceedings of the First Instructional Conference on Machine Learning, 242*, 133–142.

Rhanoui, M., Mikram, M., Yousfi, S., & Barzali, S. (2019). A CNN-BiLSTM model for document-level sentiment analysis. *Machine Learning and Knowledge Extraction, 1*(3), 832–847. https://doi.org/10.3390/make1030048

Robindro, K., Devi, S. S., Clinton, U. B., Takhellambam, L., Singh, Y. R., & Hoque, N. (2024). Hybrid distributed feature selection using particle swarm optimization-mutual information. *Data Science and Management, 7*(1), 64–73.

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. 757–762. https://doi.org/10.1109/ICMLA.2018.00120.

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *WIREs, 8*(4), e1249. https://doi.org/10.1002/widm.1249

Dos Santos, C. & Gatti, M., (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers* , pp. 69–78.

Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D. J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., & Ball, T. (2017). Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping, 38*(11), 5391–5420.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Process, 45*, 2673–2681. https://doi.org/10.1109/78.650093

Sharma, R., Sibal, R., & Sabharwal, S. (2021). Software vulnerability prioritization using vulnerability description. *International Journal of System Assurance Engineering and Management, 12*, 58–64.

Spelmen, V. S., Porkodi, R. (2018). A Review on Handling Imbalanced Data. In: International Conference on Current Trends towards Converging Technologies (ICCTCT), 1–11, https://doi.org/10.1109/ICCTCT.2018.8551020.

Stuckman, J., Walden, J., & Scandariato, R. (2017). The Effect of Dimensionality Reduction on Software Vulnerability Prediction Models. *IEEE Transactions on Reliability, 66*(1), 17–37. https://doi.org/10.1109/TR.2016.2630503

Sun, X., Ye, Z., Bo, L., Wu, X., Wei, Y., Zhang, T., & Li, B. (2023). Automatic software vulnerability assessment by extracting vulnerability elements. *Journal of Systems and Software, 204*, 111790. https://doi.org/10.1016/j.jss.2023.111790

Sun, X., Li, L., Bo, L., Wu, X., Wei, Y., & Li, B. (2024). Automatic software vulnerability classification by extracting vulnerability triggers. *Journal of Software: Evolution and Process, 36*(2), e2508. https://doi.org/10.1002/smr.2508

Sun, F. & Chu, N., (2020). Text sentiment analysis based on CNN-BiLSTM-attention model. In *2020 International Conference on Robots & Intelligent System (ICRIS),* pp. 749–752. https://doi.org/10.1109/ICRIS52159.2020.00186

Tai, K.S., Socher, R., Manning, C.D. (2015). Improved semantic representations from tree-structured long short-term memory networks. arXiv, arXiv:1503.00075. https://doi.org/10.48550/arXiv.1503.00075

Tang, G., Meng, L., Wang, H., Ren, S., Wang, Q., Yang, L., Cao, W. (2020). A Comparative Study of Neural Network Techniques for Automatic Software Vulnerability Detection. In: International Symposium on Theoretical Aspects of Software Engineering (TASE). https://doi.org/10.1109/TASE49443.2020.00010

Thekkekara, J. P., Yongchareon, S., & Liesaputra, V. (2024). An attention-based CNN-BiLSTM model for depression detection on social media text. *Expert Systems with Applications, 249*, 123834.

Vijayarani, S., Ilamathi, J., & Nithya, S. (2015). Preprocessing Techniques for Text Mining-An Overview. *International Journal of Computer Science & Communication Networks, 5*, 7–16.

Wang, P., Zhou, Y., Sun, B., Zhang, W. 2019. Intelligent Prediction of Vulnerability Severity Level Based on Text Mining and XGBboost. In: Eleventh International Conference on Advanced Computational Intelligence (ICACI). pp. 72–77, https://doi.org/10.1109/ICACI.2019.8778469.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Wohlin, C. (2007). Empirical Software Engineering: Teaching Methods and Conducting Studies. In: Empirical Software Engineering - Dagstuhl Seminar Proceedings (LNCS 4336), pp. 135–142. https://doi.org/10.1007/978-3-540-71301-2_42

Wolpert, D. H., Macready, W. G., et al. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Xue, B., Zhang, M., Browne, W. N., & Yao, X. (2016). A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation, 20*(4), 606–626. https://doi.org/10.1109/TEVC.2015.2504420

Yang, G., He, J., Lan, X., Li, T., & Fang, W. (2024). A fast dual-module hybrid high-dimensional feature selection algorithm. *Information Sciences, 681*, 121185.

Yin, W., Kann, K., Yu, M., Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. arXiv:1702.01923. https://doi.org/10.48550/arXiv.1702.01923

Yong, F., Yongcheng, L., Cheng, H., & Liang, L. (2020). FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *PLoS ONE, 15*(2), e0228439. https://doi.org/10.1371/journal.pone.0228439

Zerkane, S., Espes, D., Parc, P. L., Cuppens, F. (2016). Vulnerability analysis of software defined networking. In: International symposium on foundations and practice of security (pp. 97–116). Springer, Cham. https://doi.org/10.1007/978-3-319-51966-1_7

Zhang, Y. and Wallace, B., (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint* arXiv:1510.03820.

Ruchika Malhotra is Head of Department of Software Engineering Department Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She is Associate Dean in Industrial Research and Development, Delhi Technological University. She was awarded with prestigious Raman Fellowship for pursuing Post doctoral research in Indiana University Purdue University Indianapolis USA. She received her master's and doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She was an Assistant Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. Her h-index is 33 as reported by Google Scholar. She is reported to be in the list of world's top 2% scientists by Stanford University. She is author of book titled "Empirical Research in Software Engineering" published by CRC press and co-author of a book on Object Oriented Software Engineering published by PHI Learning. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics and the definition and validation of software metrics. She has published more than 200 research papers in international journals and conferences.

Vidushi is currently working as an Assistant Professor at Vivekananda Institute of Professional Studies and also pursuing her doctoral degree from Delhi Technological University. She completed her master's degree in Computer Science from the Jaypee Institute of Information Technology University, India. Her research interests are in software quality improvement and applications of machine learning techniques in software vulnerability prediction. She has various publications in international conferences and journals.