IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 July 2023, accepted 29 August 2023, date of publication 8 September 2023, date of current version 11 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3312613

SURVEY

A Systematic Literature Review on Software Vulnerability Prediction Models

DEEPALI BASSI AND HARDEEP SINGH
Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab 143005, India
Corresponding author: Deepali Bassi (deepalics.rsh@gndu.ac.in)

ABSTRACT The prediction of software vulnerability requires crucial awareness during the software specification, design, development, and configuration to achieve less vulnerable and secure software. Software vulnerability prediction is the process of model development that can be beneficial for the early prediction of vulnerable components at various granularity levels such as file, class, and method. Machine learning and deep learning techniques are gaining popularity in developing vulnerability prediction models. This paper performs a systematic review of primary studies from 2000 to 2022 in the literature that used machine learning and deep learning techniques for software vulnerability prediction. In addition to this, the paper understands the concept of resampling methods to handle imbalanced dataset problems; summarizes the important hyperparameter optimization methods to tune hyperparameters; explains the types of features, data pre-processing techniques, dimensionality reduction, and feature selection techniques. Furthermore, encapsulating the comparison of ML/DL techniques and highlighting the best technique is performed. The paper identifies seventy-seven research studies that use thirty-two machine learning and five deep learning techniques. Additionally, it identifies five different feature types, data pre-processing methods, thirty-seven datasets, nine data balancing techniques, twenty-six performance measures, six hyperparameter optimization methods, and the ranges of hyperparameters. Finally, guidelines for researchers to increase the productivity of software vulnerability prediction models have been illustrated in the paper.

INDEX TERMS Systematic literature review, software vulnerability prediction, machine learning, deep learning.

I. INTRODUCTION

The demand for software has increased with the advent of the technological world. Furthermore, the fourth Industrial Revolution (IR 4.0) has promoted the automation of information systems that gave way for hackers to intrude into the systems leading to financial and confidential data losses. A few examples that state the damages caused by software vulnerabilities are popular web browser plugins such as Adobe Flash Player and Oracle Java; open-source software Heartbleed, ShellShock, and Apache Commons. The browser plugins have threatened the security of millions of internet users and the open-source software has threatened thousands of companies and customers across the world. In addition to this, financial losses have also occurred of 1.7 million USD [1] due to software failure. In 2017, cybercrimes also made organizations spend 1.4 million USD and 1.3 million USD to deal with cyberattacks in 2018 [2]. There has been an exponential increase in software vulnerabilities since 2016 reported by the National Institute of Standards and Technology (NIST) [3].

Software vulnerability can be defined as an "error" that is caused during the software development life cycle by the programming mistakes of the developers. Vulnerabilities provide loopholes for attackers to invade information systems and perform malicious activities [4]. Prediction of vulnerable components at a prior stage formulates an essential step in achieving the quality and security of the software. The software vulnerability prediction (SVP) model classifies the software components as vulnerable and non-vulnerable classes. In a software security context, a vulnerability analysis system is said to be sound if it rejects all the vulnerable programs and is said to be complete if it accepts all the secure

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Destefanis.

programs. The vulnerability analysis system gives binary output and there exists a vulnerability discovery/reporting system that describes the details of discovered vulnerabilities i.e. type, location, etc.

Different approaches have been incorporated over the past years to mitigate the issues caused by software vulnerabilities. Conventional approaches include static analysis, dynamic analysis, hybrid analysis, software penetration testing, fuzz testing, and static data-flow analysis. These approaches have the drawback of high time consumption and high false-positive rates therefore machine learning (ML) and deep learning (DL) approaches have become popular [5]. The review paper [4] has categorized the research studies in the area of software vulnerability analysis and discovery based on ML and data-mining techniques into four categories namely software metrics-based SVP models, approaches on anomaly detection, pattern recognition of vulnerable code, and miscellaneous approaches. Paper [6] discusses existing approaches such as static analysis, hybrid analysis, and testing to mitigate program security vulnerabilities.

The information regarding SVP models needs to be disseminated as it can help the developers tackle the problem of software vulnerability at the early stages of software development. There exists to review works that give insight into the approaches, challenges, and open issues in the field of program security vulnerability [6]; conventional and current (ML and data-mining) approaches in the area of software vulnerability analysis and discovery [4]; encapsulation of the process of predicting software vulnerability using machine learning techniques with feature engineering, categorizing existing works into four feature types, challenges faced during feature-based ML [7]; focusing on current research issues in software vulnerability detection and reducing the gaps by presenting a taxonomy of research interests and ML methods [8]; and knowledge about data preparation challenges in the field of SVP [9].

Recent review works have focused on categorizing the works mainly based on detecting the vulnerable components, description of approaches to mitigate vulnerabilities, or the data preparation process. The current paper is motivated to unveil the knowledge about the factors that affect the efficiency of SVP models. Mainly, it revolves around three main factors i.e. the hyperparameters, feature selection (FS) criteria, and data balancing approaches. This knowledge will assist researchers in analyzing the better understanding of the ways to improve the efficacy of SVP models. This review paper aims to encapsulate, analyze, and evaluate the empirical validation of the ML, DL, resampling, FS, and hyperparameter optimization (HPO) techniques used in previous research papers. The paper reviews the research studies between 2000 to 2022.

The contributions are as follows:
- Searches the existing research works that involve ML and DL techniques in SVP models.
- Understanding the concept of data imbalance and the measures to tackle it.

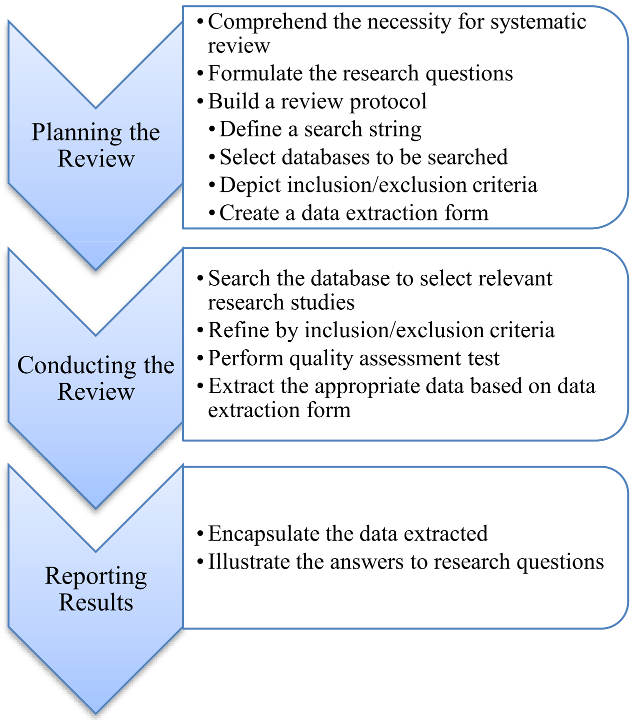

FIGURE 1. Systematic literature review process.

- Summarizing the important hyperparameters and their ranges for each ML and DL technique that can be tuned to enhance the productivity of SVP models.
- Explaining which type of features can be used in constructing SVP models and how FS methods impact the performance.
- Encapsulating the comparison of ML/ DL techniques based on different performance measures and highlighting the best technique.

The remaining paper is organized as Section II explains the methodology to conduct the review, Section III presents the results of SLR, Section IV gives the threats to validity, and Section V concludes the paper and suggests future recommendations.

II. RESEARCH METHODOLOGY

The current paper has performed a systematic literature review (SLR) based on the procedure mentioned by [10] and [11] depicted in FIGURE 1. The first stage of SLR is the Planning Stage which includes the formation of research questions and building a review protocol that focuses on research questions.

Creation of a search string, selecting a database to search primary studies, describing inclusion or exclusion criteria, and creation of data extraction form are the steps of the review protocol. The second stage is the Conducting Stage where databases are searched to collect relevant primary studies, eliminated using inclusion/exclusion criteria, performing quality assessment tests, and extracting appropriate data from final primary studies. Finally, the last stage summarizes the

TABLE 1. Research questions framed for the systematic literature review.

RQ#	Research questions	Motivation
RQ1	What are the various types of ML and DL techniques used for SVP?	To know which ML and DL techniques have been used to date.
RQ2	Which empirical validation is found for predicting vulnerabilities using ML and DL approaches mentioned in RQ1?	Evaluate the empirical evidence acquired
RQ2.1	Which feature types are used for SVP models?	To understand different types of features that can be used in constructing SVP models.
RQ2.2	Which feature extraction, reduction, or selection techniques have been used?	Identification of relevant features using appropriate feature extraction, reduction, or selection techniques
RQ2.3	Which datasets are used?	To recognize which programming languages are used for SVP models
RQ2.4	Are the datasets balanced or imbalanced?	To know the distribution of vulnerabilities in the datasets and check whether the datasets are balanced or imbalanced.
RQ2.5	What are the data balancing techniques applied?	Identify the resampling or data balancing techniques that have been applied.
RQ2.6	Which cross-validation methods have been applied?	To understand which cross-validation has been used in training and testing ML algorithms.
RQ2.7	What are the evaluation measures used?	To explore the performance metrics to evaluate the efficiency of the SVP models
RQ2.8	Which HPO methods have been applied for parameter tuning?	Identify the methods to optimize hyperparameters of ML and DL techniques
RQ2.9	What are the hyperparameters that are tuned?	To find the hyperparameters and their ranges that have been tuned.
RQ3	Which studies have shown the comparison of ML/DL techniques?	To understand which ML/DL technique suits best each dataset

TABLE 2. Framing of search string.

Category	Search Terms
Population	“software vulnerability”
Intervention	“artificial intelligence” OR “machine learning” OR “deep learning” OR “classifier” OR “prediction” OR “neural network”
Comparison	“static analysis” OR “dynamic analysis” OR “hybrid analysis” OR “fuzz-testing” OR “penetration testing” OR “code-inspections”
Outcomes	“software vulnerability” AND (“classify” OR “discover” OR “detect” OR “predict”)
Context	“software vulnerability prediction”

data extracted, illustrates the research questions, and reports the gaps and future suggestions.

A. FORMULATING RESEARCH QUESTIONS

The research questions are articulated to analyze and evaluate the empirical evidence gained from the research studies using ML and DL techniques for SVP in the literature. Table 1 defines the research questions with the motivation behind formulating them.

B. SEARCH STRATEGY

The paper uses a search strategy to identify all appropriate research papers from different digital libraries using the search strings.

1) SEARCH STRINGS

PICOC (Population, Intervention, Comparison, Outcomes, and Context) criteria suggested by [12] are used to formulate the search strings. The steps to frame search strings are as follows:

- Search terms are identified from PICOC
- Search terms that extract the studies to answer the research questions
- Identification of terms in apt titles, keywords, and abstracts
- Find synonyms, antonyms, and other spelling
- Boolean ANDs and ORs are used to identify search string

2) SOURCES SEARCHED

The current study has tried to use popular databases to search relevant studies for the survey such as IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, World Scientific, Wiley online library, and Google Scholar.

C. STUDY SELECTION PROCESS

The research studies for SLR are searched from the sources using the search string mentioned in section B.1 and section B.2. The paper restricts the search from 2000 to 2022.

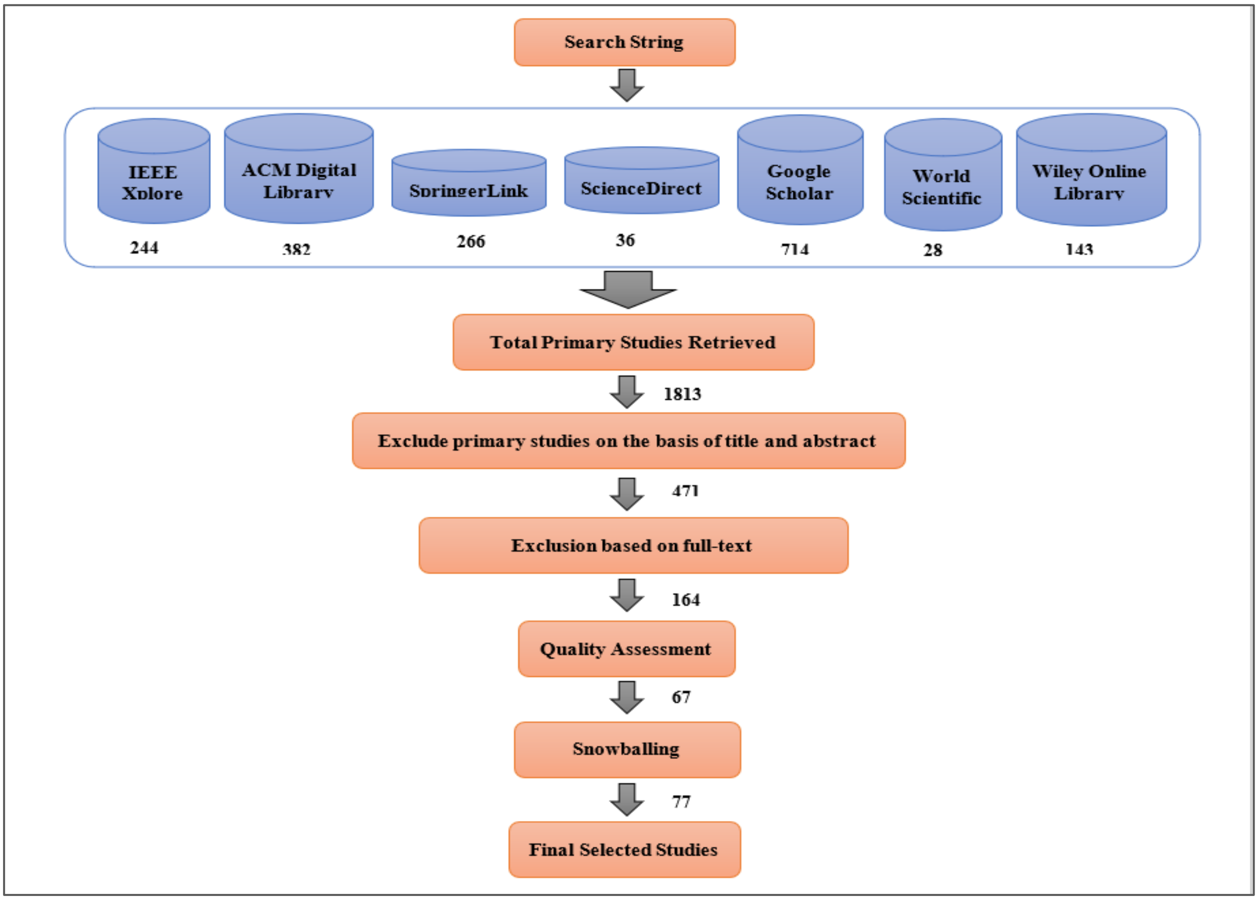

FIGURE 2. Study selection process.

FIGURE 2 represents the study selection process. Initially, on searching various databases we found 1813 articles.

Inclusion/Exclusion Criteria:

Inclusion and exclusion criteria are used for refining the collection of research studies obtained after searching the databases to figure out the most relevant study for the SLR.

Inclusion Criteria

- Studies that are related to SVP
- A study should have empirical validation of SVP
- Studies that include machine learning or deep learning techniques
- Studies that include conventional approaches are included for comparison
- Considered the studies that are published in journals and conferences

Exclusion Criteria

- Eliminating redundant studies
- Studies not available in English
- Excluding multiple versions of one study
- Full-text of the study is not available

Out of 1813 primary studies, 471 studies are included based on title and abstract. Further, we downloaded these studies for full-text review and finally shortlisted 164 studies.

D. QUALITY ASSESSMENT

We created the quality assessment questionnaire to evaluate the applicability and strength of the primary studies. These questions are considered by the suggestions in [13]. Table 3 presents the quality questions which are scored as 1 for ‘‘yes’’, 0.5 for ‘‘partly’’, and 0 for ‘‘no

The quality score will rank the papers as high ($8 \leq$ score $\leq$ 12), medium ($4 <$ score < 8), and low (score ≤ 4). 164 studies went through a quality assessment test and 67 studies were taken into consideration that have medium or high ranks. The quality score of each primary study is mentioned in Appendix A. There lies a possibility that the search strategy might skip some of the relevant studies therefore manual search such as forward and backward snowballing is performed [14].

E. SNOWBALLING

The snowballing obtains additional studies that are not obtained from searching automatically or are not present in the digital libraries. This technique extracts the relevant papers from citations or references list of the set of studies included. Further based on inclusion/exclusion and quality assessment criteria these papers are added to the final pool. The final pool contains 77 primary studies mentioned in Table 4.

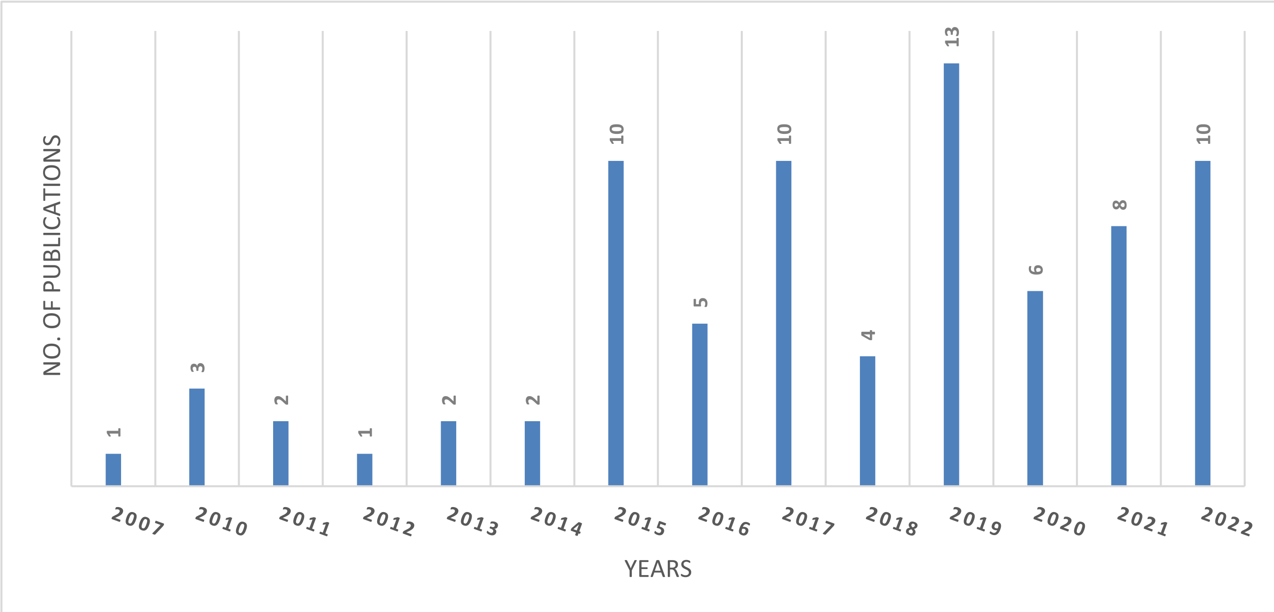

FIGURE 3. Year-wise distribution of research publications.

TABLE 3. Quality assessment questions.

Q1: Does the study belong to SVP?
Q2: Does the study apply any ML or DL techniques?
Q3: Does the study specify features of source code to apply to ML/DL techniques?
Q4: Does the study apply any feature extraction, reduction, or selection techniques?
Q5: Is the dataset clearly defined i.e. its source, distribution, and description?
Q6: Is the dataset balanced using resampling methods?
Q7: Is any statistical test applied to analyze the statistical significance?
Q8: Does cross-validation apply to the study?
Q9: Are the evaluation measures well-described?
Q10: Is cross-project or cross-version prediction applied in the study?
Q11: Have the hyperparameters tuned for ML/DL methods?
Q12: Is there any comparative analysis performed in the study?

TABLE 4. Search results.

Resource	Total results obtained	Results after inclusion/ exclusion criteria	Results after quality assessment criteria
IEEE Xplore	244	60	29
ACM Digital Library	382	41	18
SpringerLink	266	13	05
ScienceDirect	36	04	00
Google Scholar	714	41	12
World Scientific	28	00	00
Wiley online library	143	05	03
Snowballing	10	10	10
Final Pool	1823	174	77

III. RESULTS

The data is extracted from the selected studies to answer the research questions. Out of these 77 studies, 25 are published in journals, 48 are published at conferences, 2 in the symposium, and 2 in the workshop. The research studies related to SVP published in different years are depicted in FIGURE 3. The highest number of publications was observed in 2019. Around 85% of publications

TABLE 5. Software vulnerability prediction studies.

Study ID	Title and Year	Ref. No
SV1	Predicting vulnerable software components, 2007	[15]
SV2	Beyond heuristics: learning to classify vulnerabilities and predict exploits, 2010	[16]
SV3	Predicting vulnerable software components with dependency graphs,2010	[17]
SV4	Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista, 2010	[18]
SV5	An initial study on the use of execution complexity metrics as indicators of software vulnerabilities, 2011	[19]
SV6	Using SQL Hotspots in a Prioritization Heuristic for Detecting All Types of Web Application Vulnerabilities, 2011	[20]
SV7	Predicting common web application vulnerabilities from input validation and sanitization code patterns, 2012	[21]
SV8	Mining SQL injection and cross-site scripting vulnerabilities using hybrid program analysis, 2013	[22]
SV9	Using Software Structure to Predict Vulnerability Exploitation Potential, 2014	[23]
SV10	Predicting Vulnerable Components: Software Metrics vs Text Mining, 2014	[24]
SV11	Towards Cross Project Vulnerability Prediction in Open Source Web Applications, 2015	[25]
SV12	VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits, 2015	[26]
SV13	Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning, 2015	[27]
SV14	Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications, 2015	[28]
SV15	Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-Aware Perspective, 2015	[29]
SV16	Buffer Overflow Vulnerability Prediction from x86 Executables Using Static Analysis and Machine Learning, 2015	[30]
SV17	The effect of dimensionality reduction on software vulnerability prediction models, 2015	[31]
SV18	Combining software metrics and text features for vulnerable file prediction, 2015	[32]
SV19	An Empirical Investigation of Security Vulnerabilities within web applications, 2015	[33]
SV20	Text-mining based predictive model to detect XSS vulnerable files in web applications, 2015	[34]
SV21	Evaluating and comparing complexity, coupling, and a new proposed set of coupling metrics in cross-project vulnerability prediction, 2016	[35]
SV22	Is Newer Always Better?: The Case of Vulnerability Prediction Models, 2016	[36]
SV23	Experimenting Machine Learning Techniques to Predict Vulnerabilities, 2016	[37]
SV24	Early Identification of Vulnerable Software Components via Ensemble Learning, 2016	[38]
SV25	Analyzing vulnerability reproducibility for Firefox browser, 2016	[39]
SV26	Automatic feature learning for vulnerability prediction, 2017	[40]
SV27	Deep Learning Approach for Software Maintainability Metrics Prediction, 2017	[41]
SV28	Predicting Exploitation of Disclosed Software Vulnerabilities Using Open-source Data, 2017	[42]
SV29	Predicting Android application security and privacy risk with static code metrics, 2017	[43]
SV30	Predicting Vulnerable Software Components through Deep Neural Network, 2017	[44]
SV31	Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description, 2017	[45]
SV32	Evaluating micro patterns and software metrics in vulnerability prediction, 2017	[46]
SV33	Towards a software vulnerability prediction model using traceable code patterns and software metrics, 2017	[47]
SV34	When Do Changes Induce Software Vulnerabilities? 2017	[48]
SV35	Development of software vulnerability prediction web service based on ANN, 2017	[49]
SV36	Towards data-driven vulnerability prediction for requirements, 2018	[50]
SV37	Vulnerability Prediction Based on Weighted Software Network for Secure Software Building, 2018	[51]
SV38	Categorizing and Predicting Invalid Vulnerabilities on Common Vulnerabilities and Exposures, 2018	[52]
SV39	A Comparison of Nano-Patterns vs. Software Metrics in Vulnerability Prediction, 2018	[53]
SV40	A Conceptual Replication on Predicting the Severity of Software Vulnerabilities, 2019	[54]
SV41	A cost-effective strategy for software vulnerability prediction based on bellwether analysis, 2019	[55]
SV42	The importance of accounting for real-world labeling when predicting software vulnerabilities, 2019	[56]
SV43	A machine learning based approach to identify SQL injection vulnerabilities, 2019	[57]
SV44	Challenging Machine Learning Algorithms in Predicting Vulnerable JavaScript Functions, 2019	[58]
SV45	Using Software Metrics for Predicting Vulnerable Code-Components: A Study on Java and Python Open Source Projects, 2019	[59]
SV46	The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models, 2019	[60]
SV47	Predicting Vulnerable Software Components via Bellwethers, 2019	[61]
SV48	Large-Scale Empirical Studies on Effort-Aware Security Vulnerability Prediction Methods, 2019	[62]

TABLE 5. *(Continued.)* **Software vulnerability prediction studies.**

SV49	Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models, 2019	[63]
SV50	Better Security Bug Report Classification via HPO, 2019	[64]
SV51	The significant effect of parameter tuning on SVP models, 2019	[65]
SV52	Predicting web vulnerabilities in web applications based on machine learning, 2019	[66]
SV53	The effect of Bellwether analysis on software vulnerability severity prediction models. 2020	[67]
SV54	Exploitability prediction of software vulnerabilities, 2020	[68]
SV55	Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach, 2020	[69]
SV56	Suzzer: A Vulnerability-Guided Fuzzer Based on Deep Learning, 2020	[70]
SV57	Machine Learning Approach to Predict Computer Operating Systems Vulnerabilities, 2020	[71]
SV58	Vulnerability Prediction From Source Code Using Machine Learning, 2020	[72]
SV59	Empirical Studies on the impact of filter-based ranking feature selection on security vulnerability prediction, 2020	[73]
SV60	Bug Prediction Using Source Code Embedding Based on Doc2Vec, 2021	[74]
SV61	Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features, 2021	[75]
SV62	The Role of Vulnerable Software Metrics on Software Maintainability Prediction, 2021	[76]
SV63	A Comparison of Different Source Code Representation Methods for Vulnerability Prediction in Python, 2021	[77]
SV64	Efficient Feature Selection for Static Analysis Vulnerability Prediction, 2021	[78]
SV65	How to better distinguish security bug reports, 2021	[79]
SV66	Improving Vulnerability Prediction of JavaScript Functions using process metrics, 2021	[80]
SV67	V-Fuzz: Vulnerability Prediction-Assisted Evolutionary Fuzzing for Binary Programs, 2022	[81]
SV68	An Empirical Evaluation of the Usefulness of Word Embedding Techniques in Deep Learning-Based Vulnerability Prediction, 2022	[82]
SV69	Machine learning techniques for software vulnerability prediction: a comparative study, 2022	[83]
SV70	LineVul: a transformer-based line-level vulnerability prediction, 2022	[84]
SV71	Dazzle: using optimized generative adversarial networks to address security data class imbalance issue, 2022	[85]
SV72	Examining the Capacity of Text Mining and Software Metrics in Vulnerability Prediction, 2022	[86]
SV73	Software Vulnerability Prediction Using Grey Wolf-Optimized Random Forest on the Unbalanced Data Sets, 2022	[87]
SV74	Predicting input validation vulnerabilities based on minimal SSA features and machine learning, 2022	[88]
SV75	Are Source Code Metrics "Good Enough" in Predicting Security Vulnerabilities? , 2022	[89]
SV76	Optimizing hyperparameters for improvement in SVP models, 2022	[90]

have been identified after 2014 which indicates the grown interest in ML and DL. Table 5 provides the unique Study ID corresponding to each primary study selected and its reference.

A. ANSWERS TO RESEARCH QUESTIONS

RQ 1: What are the various types of ML and DL techniques used for SVP?

After reviewing the primary studies it is noticed that a large variety of ML and DL techniques have been applied to increase the productivity of SVP models. Table 6 describes various ML and DL techniques and their frequency of usage in different studies. There are 32 machine learning algorithms and 5 deep learning algorithms that have been implemented in the area of vulnerability prediction of software components. RF and SVM are used in 33 studies, LR is used in 32 studies, DT in 26 studies, NB in 30 studies, and KNN in 18 studies. Conventional machine learning algorithms have been used in most of the studies whereas ML techniques like ensemble learning, gradient boosting, adaboost, RUSBoost, and XGBoost are used in a few studies. In the case of deep learning, LSTM is used in 12 studies, GRU and DNN in 5 studies, CNN in 4 studies, and BPNN in 2 studies. Furthermore, Table 7 depicts the tools used to implement ML/DL techniques. It has been observed that Weka is the most widely used tool for implementing machine learning algorithms and Tensorflow with Keras is used widely for deep learning.

RQ 2: Which empirical validation is found for predicting vulnerabilities using ML and DL approaches mentioned in RQ 1?

This question determines the types of features used for ML/DL models, feature extraction, reduction or selection techniques, the types of datasets used and their description, data balancing techniques, training strategies applied, measures to evaluate the performance of SVP models, which hyperparameters are tuned and using which HPO method.

RQ 2.1: Which feature types are used for SVP models?

The type of feature is vital to process ML/DL algorithms. This paper classifies feature types into 5 categories code attributes, metrics (lines of code, function calls, etc.), text features (bug reports, function calls, source code imports), a combination of metrics and text features, and patterns

TABLE 6. Machine learning and deep learning algorithms.

Sr No.	Machine Learning or Deep Learning Method	Count	Study ID
1	Adaboost (AB)	6	SV42, SV46, SV48, SV57, SV65, SV77
2	Artificial Neural Network(ANN)	3	SV3, SV35, SV37
3	Average Perceptron	1	SV35
4	Bagging	4	SV14, SV20, SV48, SV70
5	Bayesian Network	2	SV3, SV37
6	Decision Tree (DT)	26	SV7, SV12, SV14, SV18, SV19, SV20, SV23, SV25, SV27, SV29, SV32, SV34, SV35, SV44, SV48, SV51, SV52, SV54, SV60, SV65, SV67, SV72, SV73, SV75, SV76, SV77
7	Density-based Cluster	1	SV21
8	Decision Forest	1	SV35
9	Decision Jungle	1	SV35
10	Ensemble Learning	1	SV24
11	Gradient Boosting	1	SV77
12	JRip	2	SV14, SV20
13	K-Means Clustering	1	SV8
14	K-Nearest Neighbor (KNN)	18	SV29, SV41, SV42, SV44, SV46, SV48, SV50, SV51, SV53, SV57, SV60, SV65, SV66, SV67, SV72, SV73, SV75, SV77
15	Linear Discriminant Analysis (LDA)	1	SV46
16	Linear Regression	3	SV44, SV60, SV67
17	Logistic Regression (LR)	32	SV3, SV5, SV6, SV8, SV12, SV14, SV16, SV19, SV21, SV23, SV25, SV35, SV36, SV39, SV41, SV42, SV44, SV45, SV48, SV50, SV53, SV54, SV55, SV57, SV60, SV65, SV66, SV67, SV72, SV75, SV76, SV77
18	Logistic Model Tree	1	SV48
19	Multilayer Perceptron (MLP)	11	SV7, SV8, SV16, SV43, SV50, SV51, SV58, SV66, SV70, SV73, SV77
20	Naïve Bayes (NB)	30	SV3, SV7, SV11, SV14, SV16, SV18, SV19, SV20, SV21, SV23, SV25, SV32, SV36, SV37, SV38, SV39, SV44, SV48, SV50, SV52, SV54, SV57, SV60, SV65, SV66, SV67, SV73, SV74, SV76, SV77
21	Random Forest (RF)	33	SV3, SV10, SV11, SV13, SV14, SV15, SV17, SV18, SV19, SV22, SV23, SV25, SV29, SV37, SV38, SV41, SV42, SV44, SV46, SV50, SV51, SV52, SV53, SV54, SV57, SV59, SV60, SV66, SV67, SV72, SV73, SV74, SV75, SV77
22	Random Subspace	1	SV48
23	Regression Tree	1	SV29
24	Regression Forest	1	SV27
25	Ridge Regression	1	SV27
26	Rotation Forest	1	SV48
27	RUSBoost	2	SV46, SV65
28	Sequential Minimum Optimization (SMO)	3	SV3, SV16, SV48
29	Stacking Classifier	1	SV76
30	Subspace Discriminant (SD)	2	SV46, SV65
31	Support Vector Machine (SVM)	33	SV1, SV2, SV4, SV9, SV11, SV12, SV14, SV19, SV24, SV27, SV28, SV29, SV31, SV33, SV35, SV36, SV37, SV39, SV41, SV44, SV45, SV46, SV51, SV55, SV60, SV65, SV67, SV70, SV72, SV73, SV74, SV75, SV77
32	XGBoost	2	SV40, SV76
33	Convolutional Neural Network (CNN)	4	SV30, SV40, SV43, SV58
34	Long Short Term Memory (LSTM)	12	SV26, SV27, SV40, SV43, SV56, SV61, SV62, SV63, SV64, SV69, SV73, SV75
35	Gradient Recurrent Unit (GRU)	5	SV10, SV63, SV69, SV73, SV75
36	Deep Neural Network (DNN)	5	SV30, SV41, SV44, SV53, SV60, SV67
37	Back Propagation Neural Network (BPNN)	2	SV70, SV71

(micro, nano). Table 8 describes the studies where the above-mentioned feature types have been used. It has been found that metrics and text-features are used in 37 and 43 studies respectively. Only two studies [46] and [73] used a

TABLE 7. Tools for implementing ML/DL techniques.

Tools	Study ID
Weka	SV3, SV6, SV7, SV8, SV10, SV11, SV[13-17], SV19, SV20, SV23, SV25, SV[32-35], SV39, SV41, SV43, SV45, SV46, SV48, SV53, SV55, SV59, SV60, SV63, SV70
Scikit-learn	SV29, SV38, SV43, SV44, SV50, SV58, SV66, SV67, SV69, SV73, SV74
Python Script	SV9, SV31, SV40, SV57, SV65, SV75, SV76
MATLAB	SV70
LibLinear	SV12
LibSVM	SV24
R Script	SV5, SV24, SV30
Azure ML Studio	SV35
Java package	SV13
Theano	SV26, SV62
Tensorflow with Keras	SV43, SV56, SV 61, SV67, SV73, SV68
PyTorch	SV71

TABLE 8. Types of features used for SVP models.

Feature Types	No. of studies	Study ID
Code Attributes	4	SV7, SV13, SV16, SV68
Metrics	37	SV[3-6], SV8, SV[10-12], SV15, SV[17-19], SV[21-23], SV25, SV27, SV29, SV32, SV33, SV35, SV37, SV41, SV42, SV45, SV46, SV48, SV51, SV52, SV55, SV59, SV61, SV63, SV67, SV73, SV76, SV77
Text-features	43	SV1, SV2, SV10, SV14, SV15, SV17, SV18, SV20, SV22, SV24, SV26, SV28, SV30, SV31, SV34, SV36, SV38, SV[40-43], SV[46-50], SV53, SV54, SV[56-60], SV62, SV[64-66], SV[68-73], SV75
Combination of metrics and text features	2	SV46, SV73
Patterns	3	SV32, SV33, SV39

TABLE 9. Feature extraction, reduction, or selection techniques used.

Techniques	Name
Feature Extraction	Bag of Words
	Word2vec
	FastText
	BERT
	GloVe
Feature Reduction	Principal Component Analysis (PCA)
	Sparse principal Component Analysis (SPCA)
	Confirmatory Factor Analysis (CFA)
	Entropy Reduction
	Linear Discriminant Analysis
	Subspace Discriminant
	Filter-based ranking Chi-squared, F-Score, Gini Ratio, InfoGain, GiniIndex, FisherScore, ReliefF
Feature Selection	Feature Subset Selection
	GainRatio method
	Information Gain and Prominent IG
	Feature Ranking using Hypothesis testing
	Point Bisseral Correlation
	Variance
	Symbiotic GA
	Symbiotic Immune Network
	Correlation Analysis Techniques (Pearson, Spearman, Kendall)
	Grey Wolf Optimization (GWO)
	Particle Swarm Optimization (PSO)
	Genetic Algorithm (GA)
	Sequential forward selection, Recursive forward elimination

TABLE 10. Datasets used in the SVP area.

Name	Programming Language	Dataset Source	Nature of Dataset (Balanced (B)/ Imbalanced (I))	Study ID
Drupal, Moodle, PHPMyAdmin	PHP	http://seam.cs.umd.edu/webvuldata	I	SV10, SV11, SV15, SV[17-19], SV34, SV35, SV41, SV46, SV48, SV51, SV52, SV59, SV61, SV74, SV77
JavaScript Dataset	JavaScript	http://www.inf.u-szeged.hu/~ferenc/papers/JSVulnerabilityDataSet/	I	SV44, SV67, SV73
Mozilla Project	C/C++	www.mozilla.org	I	SV1, SV25
OSVDB and CVE databases	Unspecified	www.sysnet.ucsd.edu/projects/exploit-learn	I	SV2
JavaScript engine used in Mozilla version 1.5 and 2.0	Java, C/C++	NA	I	SV3
Windows Vista	C/C++	www.nvd.nist.gov/vuln	I	SV4, SV38
Firefox and Wireshark	C/C++	www.mozilla.org/security/known-vulnerabilities/ www.wireshark.org/security/	I	SV5
Nine releases of Word press, blogging application, six releases of WikkaWiki	PHP	http://core.trac.wordpress.org/ticket/3937 http://wush.net/trac/wikka/ticket/293 http://core.trac.wordpress.org/ticket/2041	I	SV6
geccbblite 0.1, schoolmate 1.5.4, faqforge 1.3.2, webchess 0.9.0, utopia news pro 1.1.4, yapig 0.95b, phpmyadmin 2.6.0-pl2	PHP	http://sharlwinkhin.com/phpminer.html http://securityfocus.com	I	SV7, SV8
Phorum 5.2.18, CuteSITE 1.2.3, PHPMyAdmin 3.4.4	PHP	http://www.sourceforge.net	I	SV8
Apache HTTP Server	C/C++	www.exploit-db.com/ http://osvdb.org/ http://archive.apache.org/dist/httpd	I	SV9, SV70
66 Open Source projects	C/C++	https://github.com	I	SV12
PHPMyAdmin 3.5.0	PHP	http://www.sourceforge.net	I	SV13
PHP 9408 samples	PHP	https://github.com/stivalet/PHP-vulnerability-test-suite	I	SV14, SV20
X86 executables	C	NA	I	SV16
Apache Tomcat 6.0.35, Eclipse 3.0, OpenSCA Firefox 2.0.0.5, Linux kernel 2.6.1	Java, C/C++	NA	I	SV22
Mozilla Firefox (9 versions), Google Chrome Browsers (15 versions)	C/C++	https://eden.dei.uc.pt/~nmsa/metrics-dataset	I	SV23, SV63
FbReader, QuickSearchBox, BoardGameGeek, Contacts, Gallery2,	Java	https://sites.google.com/site/textminingandroid	I	SV24

TABLE 10. ***(Continued.)*** **Datasets used in the SVP area.**

Crosswords, browser, Deskclock, Calendar, AnkiAndroid, Mms, Connectbot, QuickSearchbox, Coolreader, Mustard, K9, Camera, Email, Keepassdroid	Java	https://sites.google.com/site/textminingandroid	I	SV26, SV62
antlr, McMMO, junit, mct, mapdb, oryx	C/C++	NA	I	SV27
1179 open-source Android apps	Java	http://androsec.rit.edu	I	SV29
BoardGameGeck, Connectbot, CoolReader, AnkiDroid	Java	https://f-droid.org/	I	SV30
The dataset was created using vulnerability data crawled from the CVE Details website	Textual data	http://www.cvedetails.com	I	SV31, SV40
Apache Tomcat, Apache Camel, Stanford Securitybench	Java	http://tomcat.apache.org/ http://camel.apache.org/ https://suif.stanford.edu	I	SV32, SV33, SV36, SV45, SV55, SV76
Apache CXF	Java	http://cxf.apache.org	I	SV33, SV39, SV45, SV55
Linux, OpenSSL, Wireshark	C/C++	www.github.com	I	SV41, SV53
Software Assurance Reference Dataset (SARD) 42212 PHP cases	PHP	https://samate.nist.gov/SARD/index.php	I	SV43
Django keystone	Python	www.cvedetails.com/cve/CVE-2019-6975/ www.cvedetails.com/cve/CVE-2018-14432/	I	SV45, SV65, SV69, SV75
Google Chrome, Mozilla, Internet Explorer	C/C++	NA	I	SV47
Windows, Mac, Cisco IOS, Linux, Internet Explorer, Safari, Firefox, Chrome	C/C++	www.cvedetails.com www.cxsecurity.com www.securitydatabase.com www.securityfocus.com	I	SV49, SV53
Chromium, Wicket, Ambari, Camel, Derby	C/C++	NA	I	SV50, SV66
LAVA-M dataset, gif2png, mpg321,tcptrace, pdf2svg, xmlwf, jhead	C/C++	NA	I	SV56
Mac, Debian, Linux, Linux kernel, Ubuntu Linux, Opensuse, Enterprise Linux, Solaris Fedora, kernel,	C/C++	NA	I	SV57
Linux kernel-rt, Windows				
Draper VDISC dataset	C/C++	https://samate.nist.gov/SRD/testsuite https://www.debian.org/ https://github.com	I	SV58, SV68
Unified Bug Database	Java	www.inf.u-szeged.hu/~ferenc/papers/UnifiedBugDataset	I	SV60

TABLE 10. *(Continued.)* **Datasets used in the SVP area.**

Big_Vul	C/C++	https://github.com/ZeoVan/MSR_20_	I	SV71
Struts2core	Java	www.github.com/palmafr/MDPIData2022	I	SV76

TABLE 11. **Data balancing techniques used.**

Data Balancing methods	Study ID
Undersampling	SV5, SV10, SV31, SV37, SV54, SV58
Random undersampling	SV23, SV44
ADASYN	SV13, SV46
Weka Spreadsubsample	SV17, SV48, SV59
SMOTE	SV25, SV42, SV46, SV50, SV52, SV66, SV74
ClassBalancerFilter WEKA	SV32, SV39, SV45, SV55
Random Oversampling	SV67, SV76
Cluster SMOTE	SV46
Borderline SMOTE	SV46

TABLE 12. **Cross-validation methods.**

Validation Methods	Study ID
K-fold Cross-validation	SV3, SV5, SV7, SV8, SV10, SV13, SV18, SV20, SV23, SV25, SV26, SV29, SV30, SV32, SV33, SV35, SV[37-39], SV41, SV43, SV45, SV46, SV48, SV55, SV57, SV[59-61], SV[65-67], SV69, SV70, SV73, SV74, SV76, SV77
Random Sample	SV12
Percentage Split (Train Test Split)	SV2, SV4, SV6, SV31, SV34, SV40, SV44, SVV58, SV64, SV71, SV72, SV75
Leave one out	SV47
Stratified Sampling	SV1

combination of metrics and text features. Three studies [32], [33], and [39] have utilized patterns, and four studies [7], [13], [16], and [68] used code attributes.

RQ 2.2: Which feature extraction, reduction, or selection techniques have been used?

Feature extraction techniques are used to extract the metrics, patterns, and text attributes from the data source as the data may be present in the raw form there. The data input given to ML/DL methods needs to be pre-processed for the execution of the experiment. Some studies have included various dimensionality reduction and selection techniques to prioritize the important and dominant features thereby optimizing the SVP models. Principal component analysis (PCA) is a good technique for identifying linearly uncorrelated dimensions in large datasets with potentially many inter-correlated features applied in [41] and [43]. PCA is applied after the min-max normalization of the dataset in [22]. Min-max normalization allows our predictors to operate in a standardized data space rather than a raw data space.

Researchers used a variety of feature selection strategies to extract the most significant characteristics from a vast feature vector. It removes irrelevant and noisy features from prediction models to improve their performance. The effect of dimensionality reduction approaches (feature selection, principal component analysis, and confirmatory factor synthesis) on the output of SVP models was examined by Stuckman et al. [31]. Additionally, rather than within-project prediction, dimensionality reduction strategies fared better in cross-project prediction. The [34] study used Information Gain feature selection approaches to extract notable features and build the ProminentIG (F3) feature set. Ranking features using the Wilcoxon rank sum test is employed by [44]. The research study [47] used Point-Biserial Correlation among the nano-patterns and metrics in vulnerable and neutral methods. The goal is to identify and use strongly correlated patterns and metrics in vulnerable and neutral code to develop a prediction model. In [55], bellwether analysis, a novel method is suggested for locating and choosing an excellent subset of data to use as the training set to increase prediction accuracy. Linear discriminant analysis (LDA) and subspace discriminant (SD) are used in [60] and [67]. To improve class separability, LDA projects a dataset into a lower dimensional feature space. The LDA algorithm consists of three key phases. Calculating how easily various classes can be separated is the first step. Calculated are the differences between class attributes. The distance between the mean and samples of each class is calculated in the subsequent step. The third phase entails creating the lower dimensional space that maximizes the variance between classes and minimizes the variance within classes. A linear classifier can be created using the resulting dataset. The case of unequal intraclass frequencies is simply handled by LDA.

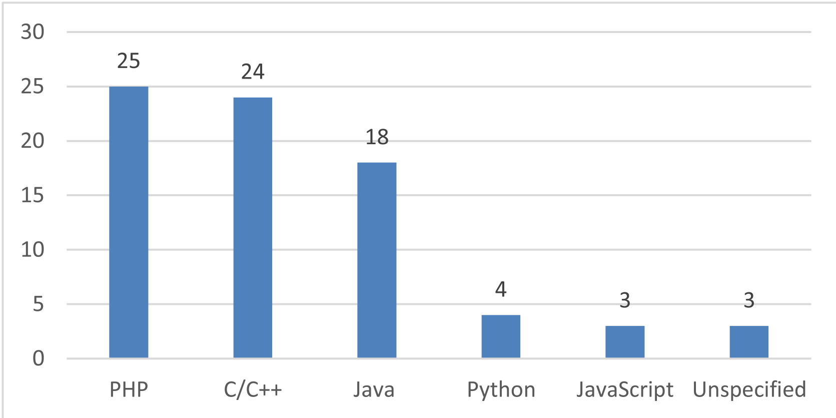

FIGURE 4. Count of studies for each programming language.

		Predicted values		
		Positive	**Negative**	Totals
Actual Values	**Positive**	TP	FN	**P = (TP + FN) = Actual Total Positives**
	Negative	FP	TN	**N = (FP + TN) = Actual Total Negatives**
	Totals	Predicted Total Positives	Predicted Total Negatives	

FIGURE 5. Confusion matrix.

TABLE 13. Training strategy.

Cross-project	SV10, SV11, SV13, SV15, SV17, SV19, SV21, SV26, SV48, SV62, SV76
Cross-version	SV23, SV26, SV76
Cross-metric	SV17

An ensemble classification technique called subspace discriminant (SD) makes use of linear discriminant classifiers. The random subspace process, also known as feature bagging, is used in the subspace discriminant classifier to lessen the correlation between estimators. The random subspace approach is comparable to bagging, however, it differs from bagging in that each learner receives a replacement when the features are randomly subsampled. It is possible to find students who have specialized knowledge of the various feature sets. Table 9 shows various techniques used for feature representation, extraction, reduction, and selection. Study [72] assessed variance for each column and identified the main components to show the most significant features and exclude the least significant elements. This allows to reduce the array size from 1533 to 250, which benefits the ML model's training process. The experimental research shows that this procedure cuts training time by roughly 68%. In [73], it is mentioned that SVP data sets frequently include several features, which leads to the dimensionality curse. Since other forms of feature selection methods have a high computational cost, the focus of this paper is on the effect of filter-based ranking feature selection (FRFS) approaches on SVP. Final results demonstrate that, as compared to state-of-the-art baselines, utilizing FRFS can enhance SVP performance, given the comparable cost of code inspection.

Study [75] used the SYMbiotic genetic algorithm with the dominance mechanism for phenotyping the dominant-feature representations, which were then fed into the deep learning framework using LSTM and GRU RNNs models. The results revealed that the proposed method (GRU-SYMbiotic GA-II) enhanced vulnerability prediction, indicating improved software quality. Reference [77] proposes a novel model for predicting software maintainability and testing it using vulnerability metrics. The proposed method was utilized to discover critical vulnerable software metrics that aid in improving software maintainability accuracy. It implements

TABLE 14. Evaluation measures used in the study.

Performance Metric	Formula	Study ID
Accuracy	$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$	SV3, SV7, SV13, SV14, SV16, SV[23-25], SV27, SV28, SV30, SV31, SV37, SV43, SV48, SV49, SV52, SV54, SV56, SV57, SV61, SV[63-65], SV[67-69], SV73, SV75, SV76
The area under the ROC curve (AUC)	$AUC = (1 + TPrate - FPrate)/2$	SV35, SV38, SV46, SV62, SV75, SV77
F-Measure/ F1-Score	$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$	SV3, SV11, SV14, SV[17-19], SV23, SV25, SV26, SV28, SV[31-34], SV[39-41], SV43, SV44, SV46, SV[52-54], SV57, SV58, SV[60-62]
Precision/ Correctness	$\text{Precision} = \frac{TP}{TP + FP}$	SV1, SV[3-8], SV11, SV12, SV14, SV16, SV19, SV21, SV[23-26], SV[28-34], SV36, SV37, SV[39-47], SV[51-59], SV61, SV62, SV64, SV66, SV67, SV69, SV[71-76]
Recall/ Sensitivity/ True positive rate (TPR)	$TPR = \frac{TP}{TP + FN} = \frac{TP}{P}$	SV1, SV[3-6], SV8, SV[10-14], SV16, SV17, SV19, SV[21-26], SV[28-34], SV36, SV37, SV[39-47], SV[50-59], SV62, SV[64-67], SV69, SV[71-76]
False Positive Rate (FPR)	$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}$	SV20, SV21, SV25, SV37, SV39, SV45, SV[54-56], SV66, SV72
File Inspection Ratio(FI)	$FI = \frac{TP + FP}{TP + TN + FP + FN}$	SV5, SV10, SV17, SV22, SV59
Specificity (TNR)	$\text{Specificity} = \frac{TN}{FP + TN}$	SV20, SV46, SV65
False negative rate (FNR)	$FNR = \frac{FN}{FN + TP}$	SV20, SV33, SV39, SV54, SV55
Probability of False Alarm (pf)	$pf = \frac{FP}{FP + TN}$	SV5, SV7, SV8, SV13, SV16
File Inspection Reduction(FIR)	$FIR = \frac{recall - FI}{recall}$	SV5
LOC Inspection Reduction (LIR)	$LI = \frac{\text{Lines of code tobe inspected}}{\text{Total lines of code}}$ $PV = \frac{\text{Files predicted as vulnerable}}{\text{Total no of vulnerabilities}}$ $FIR = \frac{PV - LI}{PV}$	SV5
Effectiveness Ratio	$ER@20\% = \frac{\text{VulPredictor (20\%)}}{\text{Perfect (20\%)}}$ $= \frac{\text{No. of files in first 20\% of ranked list by Vulpredictor}}{\text{No. of files in first 20\% ranked list by perfect technique}}$	SV18
Fβ-measure	$F\beta - measure = (1 + \beta^2) * \frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$	SV21, SV36, SV54, SV61, SV73
Informedness	$\text{Informedness} = \frac{TP}{TP + FN} - \frac{FP}{TN + FP}$	SV23
Markedness	$\text{Markedness} = \frac{TP}{TP + FP} - \frac{FN}{TN + FN}$	SV23

TABLE 14. ***(Continued.)*** **Evaluation measures used in the study.**

Mean Absolute Error (MAE)	Absolute Error (AE)= |actual – predicted| $\text{Relative Error(RE)} = \frac{\text{AE}}{\text{actual}}$ $\text{Squared Error (SE)} = (\text{actual} - \text{predicted})^2$ $\text{MAE} = \frac{1}{n}.\sum_i \text{AE}_i$	SV40, SV47, SV54, SV70
Root Mean Squared Error (RMSE)	$\text{RMSE} = \sqrt{\frac{1}{n}.\sum_i \text{SE}_i}$	SV27, SV40, SV70
Mean Absolute Percentage Error(MAPE)	$\text{MAPE} = 100.\frac{1}{n}.\sum_i \text{AE}_i$	SV27, SV40
Mean Magnitude of Relative Error (MMRE)	$\text{MNRE} = \frac{1}{n}.\sum_i \text{RE}_i$	SV40
Median Magnitude (MdMRE)	Median of REs	SV40
Matthews Correlation Coefficient (MCC)	$\text{MCC} = \frac{(\text{TP.TN}) - (\text{FP.FN})}{\sqrt{(\text{TP}+\text{FP}).(\text{TP}+\text{FN}).(\text{TN}+\text{FP}).(\text{TN}+\text{FN})}}$	SV42
Resolution Effort Ratio	$\text{Resolution Effort Ratio} = \frac{\text{No. of components needing inspection}}{\text{No. of components}}$	SV42
$\mathbf{P_{opt}}$	$\text{P}_{\text{opt}} = 1 - \frac{\text{area(optimal)} - \text{area (m)}}{\text{area(optimal)} - \text{area(worst)}}$ where, area(m), area(optimal), and area(worst) are the area under the curve corresponding to a proposed prediction method, the optimal method, and the worst method, respectively.	SV48
Top-k accuracy	Among the top-K samples, the number of samples labeled as “vulnerable” is v^. $\text{Top} - \text{k accuracy} = \frac{v^{\wedge}}{K}$	SV68
G-measure	$\text{G} - \text{measure} = \sqrt{\text{Sensitivity} * \text{Specificity}}$	SV72

a novel framework symbiotic immune network based on deep learning to improve robustness to forecast software maintainability. In [78], an empirical study in which a comparison of three-word embedding-based code representation methods in the context of vulnerability prediction in Python code is presented. These natural language processing algorithms - word2vec, fastText, and BERT - are commonly used in practice to represent source code as numeric vectors and perform SE tasks. The results show that all text representation methods are appropriate for code representation in this task, but the BERT model is the most promising because it is the least time-consuming, and the LSTM model based on it achieved the best overall accuracy (93.8%) in predicting source code vulnerabilities. Reference [79] look at features generated by the SonarQube and CCCC tools to see which ones might be used to predict software vulnerabilities. It evaluates the applicability of thirty-three different features for training thirteen different machine learning algorithms to construct vulnerability predictors and determine the most relevant features for training. The evaluation is based on a thorough feature selection process based on feature correlation analysis (Pearson, Spearman, and Kendall), as well as four well-known feature selection techniques (information gain ratio, Gini decrease index, information gain, χ^2 ranking). BERT with self-attention mechanism and CodeBERT are used in [85]. In [87], a Bag of words and sequences of text tokens is used for feature extraction and utilizes point-biserial correlation to rank features according to the correlation among them. Metaheuristic techniques like the grey wolf, particle swarm optimization, and genetic algorithm are applied in [88] for feature selection. In [89], This study presents a method for anticipating vulnerable files against input validation flaws. Transforming the programs first into intermediate representations (MSSA form) removes unnecessary instructions in the SSA form; it also illustrates the vulnerabilities with their accompanying clear dependence links.

RQ 2.3: Which datasets are used?

Table 10 describes the datasets used in the primary studies of SVP. It mentions the name of the dataset, the programming language of the dataset, the dataset source, the nature of the dataset, and the primary studies that use it. There are 37 different datasets used in the 77 primary studies. FIGURE 4 describes the programming languages used in

TABLE 15. **HPO methods and the hyperparameters that are tuned.**

HPO methods	Hyperparameters tuned	Study ID
Manual	SVM (kernel: Linear)	SV2
	CNN and LSTM (Learning_rate=0.001, weight decay=0.00001, loss(classification)=cross entropy loss, loss(regression)=mean squared error, optimizer=Adam	SV40
	RF (n_estimators=[10,50,100,150,200,250])	SV51
	Batch_size = 128, Hidden layer units=128, Hidden layer depth =2	SV56
	Hidden Layer [5,10,20,50,100]	SV58
	NB(Vector size=75. Window size=4) Linear Regression (Vector size=150, window size=8) Logistic Regression (vector size=150, window size=8, Liblinear solver, regularization wt (c)=2.0, tolerance value 0.001) Decision Tree (vector size=150, window size=12, max depth=10, gini splitting criterion) RF (vector size=150, window size=4, max depth=10, entropy, no of trees=100) CDNNC (vector size=150, window size=8, 5 layers, 200 neurons, learning_rate=0.05, binary cross-entropy, 10 epochs, batch size=100) SDNNC (vector size=75, window size=8, 5 layers, 250 neurons, 0.0005 beta value) KNN (vector size=75, window size=8, k=18, uniform distance weights) SVM (Radial Basis Function, gamma value=0.02, c=2.6)	SV59
	Momentum=0.2, learning_rate=0.3	SV61
	No. of neurons, no. of epochs	SV64
	RNN (No. of layers =3, No. of hidden layers =1(LSTM, GRU, BiLSTM), Embedding Size=300, weight initialization technique= Glorot uniform, learning_rate=0.01, gradient descent optimizer= Adam, Batch size=64, Activation function= ReLu, Output activation function= sigmoid, Loss function=Binary cross-entropy, Overfitting prevention dropout=0.3, Max epochs=100, Early Stopping patience=10) CNN (No. of layers =3, No. of hidden layers =1(1D CNN), Embedding Size=300, kernel size=5, weight initialization technique= Glorot uniform, learning_rate=0.01, gradient descent optimizer= Adam, Batch size=64, Activation function= ReLu, Output activation function= sigmoid, Loss function=Binary cross-entropy, Max epochs=100, Early Stopping patience=10)	SV69
	Feed forward Backpropagation Neural Network (Hidden layer size=10) Cascade forward Backpropagation Neural Network (Hidden layer size=10) MLP (No of hidden layers =3, activation function= sigmoid) SVM (C=1.0, batch_size=100) Bagging (Batch bag=100) M5Rules (Batch size=100, minimum no of mistakes=4.0)	SV70
	Learning_rate = 2e-5	SV71
	Batch size= 16,32,64,128 Learning rate for generator and discriminator =[0.0005, 0.001, 0.005, 0.01, 0.05, 0.1] Optimizer for generator and discriminator= [Adadelta, Adagrad, Adam, Adamax. NAdam, RMSprop, SGD] Activation function for generator and discriminator= [elu, relu, selu, sigmoid, softmax, tanh, hard_sigmoid, softplus, leakyReLu] No of Epochs=quniform(5,20,1) Generator and Discriminator layer normalization= True, False	SV72
	LSTM, BiLSTM, GRU, BiGRU (batch size=256, no. of LSTM neurons=128, output dimension=256, learning rate=0.01, sequence length =350)	SV75
	LR solvers XGBoost (No. of estimators=500, objective=binary:logistic-vulnerability prediction, multi-softmax-sensitivity and title prediction)	SV76
LibLinear	Linear SVM (regularization parameter c =1, class weight w = 100)	SV12

TABLE 15. ***(Continued.)*** **HPO methods and the hyperparameters that are tuned.**

RMSProp	LSTM (learning_rate=0.02, smoothing hyperparameters ρ=0.99, ϵ=1e-7, dropout rate=0.5) RF (no of trees, max depth, max no. of features per tree)	SV26
	Learning_rate=0.02, ρ=0.99, ϵ=1e-7, dropout rate=0.5	SV62
Sensitive Analysis of CNN parameters	No of training iterations=150, Batch_size=32, Dimensionality of output feature vectors= No of filters N=128, Dimensionality of word embeddings=300	SV32
Grid Search	Unspecified	SV44, SV67
	RF (n_estimators[10,150], max_leaf_nodes=[2,50], min_samples_leaf=[1,20], max_features=[0.01,1], min_sample_split=[2,20], max_depth[1,10]) LR (C=[1.0, 10.0], max_iter=[50,200], verbose=[0,10]) KNN (leaf_size=[10,100], n_neighbors=[1,10]) MLP (alpha=[0.0001,0.001], learning_rate_init=[0.001,0.01], power_t=[0.1,1], max_iter[50,300], momentum=[0.1,1], n_iter_no_change) NB (var_smoothing[0.0,1.0]) SMOTE (k[1,20], m[50,400], r [1,6])	SV50, SV66
	RF (No. of trees) KNN (No. of neighbors) DNN (Hidden layers)	SV73
Optuna	RF(n_estimators=[10,150], max_depth[5,50], max_features[0.01,1.0], criterion [gini, entropy]) SVM (C [1e-6,100.0], kernel [linear, poly, rbf, sigmoid] KNN (n_neighbors[1-20], leaf_size[10-100], weights[uniform, distance]) AB (n_estimators[50-500], learning_rate[0.01, 2.0], algorithm [SAMME, SAMME.R]) GNB (var_smoothing [0.0, 1.0]) DT (max_depth[1-10], max_features[0.01,1.0], criterion [gini, entropy]) LR (C[1.0-30.0], max_iter[50,200], verbose[0-10]) MLP (alpha [0.0001-0.001], learning_rate_init[0.001, 0.01], power_t[0.1-1], max_iter[50-300], momentum[0.1,1], n_iter_no_change[1-100])	SV77

different primary studies. PHP language-based dataset is used in 25 studies, C/C++ based dataset in 24 studies, Java in 18 studies, Python in 4 studies, JavaScript in 3 studies, and unspecified programming language in 3 studies. It is found that the PHP dataset (Drupal, Moodle, and PHPMyAdmin) is the highly utilized dataset i.e. in 17 studies.

RQ 2.4: Are the datasets balanced or imbalanced?

It has been observed that all the datasets used are imbalanced which is the open research problem that is catered in the next research question.

RQ 2.5: What are the data balancing techniques applied?

Software vulnerability datasets have the problem of imbalanced datasets which can be tackled by data balancing techniques (refer to Table 11). Table 11 describes various resampling methods in different primary studies. Out of 77, twenty-five primary studies have used data balancing techniques. SMOTE is maximum used resampling method in 7 studies followed by undersampling i.e. in 6 studies. To balance the data, [19], [45], [51], and [72] used under-sampling by eliminating randomly selected majority class data until the numbers of data instances in the majority and minority classes were equal. Research study [24], [31], [62], and [73] uses Weka SpreadSubsample unsupervised filter to implement undersampling. The examined systems in [37] and [58] make use of the Random Undersampling (RU) balancing mechanism, which eliminates randomly selected majority class data. The chosen percentage is maintained until the number of data instances in the majority class and the number of instances in the minority class are equal. In [27] and [60], ADASYN (adaptive synthetic oversampling) is used, which reduces the bias caused by the class imbalance problem by generating synthetic, false data for the minority class instances to balance the (unbalanced) data. It can be easily implemented as an additional data preprocessing step because it does not call for the modification of standard classifiers. In [39], [56], [60], [64], [66], [80], and [88] research studies SMOTE (synthetic minority oversampling technique) is the most popular technique for creating synthetic samples. Instead of reproducing the current samples, it makes new ones. It handles oversampling by working in feature space rather than data space. Along the line segments connecting the samples of the minority class's k-nearest neighbors, the synthetic instances are introduced. The K-nearest neighbors are selected at random. Decision boundaries become more strict and generalized with this method. Additionally, it avoids overfitting. Cluster SMOTE and Borderline SMOTE are enhanced versions of SMOTE used in [46]. In cluster SMOTE, data is clustered first and then SMOTE is applied to each cluster. Borderline SMOTE generates and adds new data from borderline samples.

TABLE 16. Comparison of ML/DL techniques.

Study ID	ML/DL techniques compared	Best-performing ML/DL techniques
SV3	BN, NB, NN, RF, SMO	BN
SV7	NB, C4.5, MLP	C4.5 and MLP
SV8	MLP, LR, K-means cluster	LR
SV10	ELM and Multi-SVM	ELM
SV11	NB, LR, SVM, J48, RF	RF
SV14	SVM, NB, Bagging, RF, J48, JRip	Bagging
SV18	VULPREDICTOR, $RF_{metrics}$, $NB_{metrics}$, $DT_{metrics}$, RF_{text}, NB_{text}, CNB_{text}	VULPREDICTOR
SV19	NB, LR, SVM, J48, RF	RF
SV20	NB, JRip, J48, RTree, Bagging	J48
SV21	NB, Classification Clustering, Threshold Selector, Random Tree	NB and Classification Clustering
SV23	Logistic Regression, Bayesian Network, DT, RF, NB	DT, LR
SV25	RF, LR, DT, NB	RF
SV27	Ridge regression, DT, Quantile Regression Forest, SVM, PCA, LSTM	LSTM
SV29	CART, KNN, r-SVM, RF	r-SVM
SV31	(TF-IDF+SVM), Word Embedding + SVM, Word Embedding +2 layer CNN, Word Embedding + CNN with LSTM, Word Embedding + 1-layer CNN	Word embedding + 1-layer CNN
SV33	LR and SVM	LR
SV35	Average Perceptron, Bayes Machine, DT, Decision Forest, Decision Jungle, Deep SVM, LR, Neural Network, SVM	NN
SV37	BN, NB, NN, RF, SVM	NN
SV39	NB, LR, SVM	SVM
SV40	CNN, LSTM, XGBoost, SVM	CNN and LSTM
SV43	DT, RF, SVM, LR, MLP, RNN, LSTM, CNN	SVM, MLP
SV44	DNN_S, DNN_C, RF, KNN, Linear Regression, LR, SVM, DT, Bayes	KNN
SV45	SVM, LR	LR
SV50	LR, RF, NB, MLP, KNN	LR
SV52	J48, RF	RF
SV53	CNN, LSTM, XGBOOST, SVM, Word Embedding + 1-layer CNN, LR, DNN, RF, KNN	RF
SV54	DT, RF, NB, LR, SVM	DT, RF
SV55	LR, SVM	SVM
SV57	LR, KNN, GNB, RF, AB	RF
SV60	Bayes, Linear regression, LR, DT, RF, CDNNC, SDNNC, KNN	SDNNC
SV61	SGA, Symbiotic GA-I, Symbiotic GA-II, LSTM-SGA, LSTM-Symbiotic GA-I, LSTM-Symbiotic GA-II, GRU-SGA, GRU-Symbiotic GA-I, GRU-Symbiotic GA-II, ANN-GA	GRU-Symbiotic GA-I
SV65	DT, KNN, LR, NB, SVM, Boosted Trees, Bagged Trees, SD, RUSBoost	Bagged Trees
SV67	KNN, DT, RFC, SVM, CDNN, SDNN, LR, Linear Regression, NB	RFC
SV69	LSTM, BiLSTM, GRU, CNN	CNN
SV72	KNN, LR, DT, RF, SVM	SVM
SV73	KNN, RF, DT, SVM, NB, ANN	RF
SV74	SVM, NB, GB, RF, GW-RF, PSO-RF, GA-RF	GW-RF, PSO-RF
SV75	DT, KNN, RF, LR, SVM, LSTM, BiLSTM, GRU, BiGRU	LSTM
SV76	DT, LR, NB, XGB	XGB
SV77	DT, RF, SVM, KNN, AB, GNB, LR, MP	GNB

Weka ClassBalancerFilter is used in [46], [53], [59], and [69]. The instances are reweighted by this filter so that the total weight for each category is the same. References [81] and [90] uses random oversampling.

RQ 2.6: Which cross-validation methods have been applied?

Table 12 shows the cross-validation methods used in various primary studies. K-fold cross-validation have been highly used. Table 13 describes the studies that have incorporated the cross-project, cross-version, and cross-metric training strategy. Such training strategies are different from within project prediction and train the ML/DL method using one project/metric/version and test on another project/metric/version.

RQ 2.7: What are the evaluation measures used?

Evaluation metrics are used to measure the performance of SVP models. The studies have used around 26 different performance measures described in Table 14. FIGURE 5 shows the confusion matrix that is used to calculate most of the performance metrics. Accuracy, F1-score, precision, and recall are widely used.

RQ 2.8: Which HPO methods have been applied for parameter tuning?

Hyperparameter tuning is essential to increase the productivity of SVP models [64], [65], [80], and [91]. There exist different HPO methods to tune the hyperparameters such as Grid search, Optuna, RMSprop, LibLinear, sensitive

TABLE 17. Quality assessment score.

Study ID	Ref	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Score	Rank
SV1	[15]	1	1	1	0	1	0	1	0.5	1	0	0	0	6.5	Medium
SV2	[16]	1	1	1	1	1	0.5	1	0	1	0	1	1	9.5	High
SV3	[17]	1	1	1	0	1	0	1	0.5	1	0	0	1	7.5	Medium
SV4	[18]	1	1	1	0	1	0	1	0	1	0	0	0	6	Medium
SV5	[19]	1	1	1	0	1	1	1	1	1	0	0	1	10	High
SV6	[20]	1	1	1	0	1	0	1	1	1	0	0	1	8	High
SV7	[21]	1	1	1	1	1	0	1	1	1	0	0	1	9	High
SV8	[22]	1	1	1	1	1	0	1	1	1	0	1	1	10	High
SV9	[23]	1	1	1	0	1	0	0	0	1	0	0	0	5	Medium
SV10	[24]	1	1	1	0	1	1	1	1	1	1	0	1	10	High
SV11	[25]	1	1	1	0	1	0	1	0	1	1	0	1	8	High
SV12	[26]	1	1	1	0	1	0	1	1	1	0	1	1	9	High
SV13	[27]	1	1	1	0	1	1	0	1	1	1	0	0	8	High
SV14	[28]	1	1	1	0	1	0	0	1	1	0	0	1	7	Medium
SV15	[29]	1	1	1	0	1	0	1	1	1	1	0	1	9	High
SV16	[30]	1	1	1	0	1	0	0	1	0	0	0	0	5	Medium
SV17	[31]	1	1	1	1	1	1	1	1	1	1	0	1	11	High
SV18	[32]	1	1	1	0	1	0	0	1	1	0	0	1	7	Medium
SV19	[33]	1	1	1	0	1	0	1	0	1	1	0	1	8	High
SV20	[34]	1	1	1	1	1	0	0	1	1	0	0	1	8	High
SV21	[35]	1	1	1	0	1	0	0	0	1	1	0	1	7	Medium
SV22	[36]	1	1	1	0	1	0	0	1	0	0	0	1	6	Medium
SV23	[37]	1	1	1	0	1	1	0	1	1	1	0	1	9	High
SV24	[38]	1	1	1	0	1	0	0	1	1	0	0	0	6	Medium
SV25	[39]	1	1	1	0	1	1	0	1	1	0	0	1	8	High
SV26	[40]	1	1	1	1	1	0	0	1	1	1	1	1	10	High
SV27	[41]	1	1	1	0	1	1	0	0	1	0	1	1	8	High
SV28	[42]	1	1	1	0	1	0	0	1	1	0	0	1	7	Medium
SV29	[43]	1	1	1	1	1	1	0	1	1	0	0	1	9	High
SV30	[44]	1	1	1	1	1	0	1	1	1	0	0	1	9	High
SV31	[45]	1	1	1	1	1	1	0	1	1	1	0	1	10	High
SV32	[46]	1	1	1	1	0	1	0	1	1	0	0	1	8	High
SV33	[47]	1	1	1	1	1	0	1	1	1	0	0	1	9	High
SV34	[48]	1	1	1	1	1	0	0	1	1	0	0	1	8	High
SV35	[49]	1	1	1	0	1	0	0	1	1	0	0	1	7	Medium
SV36	[50]	1	1	1	0	1	0	0	0	1	0	0	0	5	Medium
SV37	[51]	1	1	1	0	1	0	1	1	1	0	0	1	8	High
SV38	[52]	1	1	1	1	1	0	1	1	1	0	0	0	8	High
SV39	[53]	1	1	1	0	1	1	1	1	1	0	0	1	9	High
SV40	[54]	1	1	1	1	1	0	1	1	1	0	1	1	10	High
SV41	[55]	1	1	1	1	1	0	0	1	1	0	0	0	7	Medium
SV42	[56]	1	1	1	0	1	1	1	1	1	0	0	1	9	High
SV43	[57]	1	1	1	1	1	0	0	1	1	0	0	1	8	High
SV44	[58]	1	1	1	0	1	1	0	1	1	0	1	1	9	High
SV45	[59]	1	1	1	0	1	1	0	1	1	0	0	1	8	High
SV46	[60]	1	1	1	1	1	1	1	1	1	0	0	1	10	High
SV47	[61]	1	1	1	0	1	0	1	1	1	0	0	1	8	High
SV48	[62]	1	1	1	0	1	1	1	1	1	1	1	1	11	High
SV49	[63]	1	1	1	0	1	0	1	1	1	0	1	1	9	High
SV50	[64]	1	1	1	1	1	1	1	1	1	0	1	1	11	High
SV51	[65]	1	1	1	0	1	0	0	1	1	0	1	1	8	High
SV52	[66]	1	1	1	0	1	1	0	1	1	0	0	1	8	High

TABLE 17. *(Continued.)* **Quality assessment score.**

SV53	[67]	1	1	1	0	1	0	1	1	1	0	0	1	8	High
SV54	[68]	1	1	1	0	1	1	1	1	1	0	0	1	9	High
SV55	[69]	1	1	1	1	1	1	1	1	1	0	0	1	10	High
SV56	[70]	1	1	1	0	1	0	0	0	1	0	1	1	7	Medium
SV57	[71]	1	1	1	0	1	0	0	1	1	0	0	1	7	Medium
SV58	[72]	1	1	1	1	1	1	0	1	1	0	1	1	10	High
SV59	[73]	1	1	1	1	1	1	1	1	1	0	0	0	9	High
SV60	[74]	1	1	1	0	1	1	0	1	1	0	1	1	9	High
SV61	[75]	1	1	1	1	1	0	0	1	1	0	1	1	9	High
SV62	[76]	1	1	1	1	1	0	0	0	1	0	0	1	7	Medium
SV63	[77]	1	1	1	1	1	1	0	1	1	0	1	1	10	High
SV64	[78]	1	1	1	1	1	0	1	1	1	0	0	1	9	High
SV65	[79]	1	1	1	0	1	1	1	1	1	0	1	1	10	High
SV66	[80]	1	1	1	0	1	1	1	1	1	0	1	1	10	High
SV67	[81]	1	1	1	1	1	0	0	0	1	0	0	1	7	Medium
SV68	[82]	1	1	1	0	1	0	0	1	0	0	1	1	7	Medium
SV69	[83]	1	1	1	0	1	0	1	1	1	0	1	1	9	High
SV70	[84]	1	1	1	1	1	0	0	1	1	0	1	1	9	High
SV71	[85]	1	1	1	0	1	1	0	1	1	0	1	1	9	High
SV72	[86]	1	1	1	1	1	1	1	1	1	0	1	1	11	High
SV73	[87]	1	1	1	1	1	1	1	1	1	0	0	1	10	High
SV74	[88]	1	1	1	1	1	1	1	1	1	0	1	1	11	High
SV75	[89]	1	1	1	1	1	1	0	1	1	1	1	1	11	High
SV76	[90]	1	1	1	0	1	0	1	1	1	0	1	1	9	High

analysis, and manual. Out of 77 primary studies, 22 studies have applied HPO. Researchers have to check the appropriate HPO methods which take less computational time and the suitable hyperparameters that need to be tuned as per their requirements. This study gave them insights into which hyperparameters and methods that can be applied to machine learning algorithms to optimize SVP models.

RQ 2.9: What are the hyperparameters that are tuned?

Table 15 explains the hyperparameters that are tuned in various studies with their hyperparameter values or ranges. DL techniques have mostly tuned the hyperparameters manually.

RQ 3: Which studies have shown the comparison of ML/DL techniques?

This paper has collected 77 studies that have used ML/DL techniques in the SVP area. Forty studies have shown the comparison among various ML/DL methods. Table 16 shows which ML/DL are compared and stated the highest performance ML/DL technique. The paper has fetched the results based on F1-Score, AUC, precision, and recall to show the best-performing ML/DL techniques. It has been observed that RF is used in most of the studies and found to perform the best followed by SVM and LR.

IV. THREATS TO VALIDITY

There are a few threats that affect the validity of SLR. The first threat can be the inclusion of all possible relevant studies. Although we have tried to include all the studies that use ML/DL techniques in the software prediction area, there can be some studies that have been missed. Secondly, the quality assessment criteria and data extraction process need thorough investigation. If not done properly, there can be chances of missing studies. Thirdly, we have extracted the data as given in the previous studies and our motive is to unveil the challenges, problems, and considerations hence the data extracted by us might not be exhaustive.

V. CONCLUSION AND FUTURE GUIDELINES

The importance of ML and DL techniques has gained interest in developing software vulnerability prediction models. Researchers have come up with a wide variety of ML and DL approaches that can be used to predict vulnerable software components. These studies reveal various challenges and issues that need to be approached to understand the SVP models. Hence, there was a need to systemize the knowledge of available literature to uncover the challenges faced in developing the SVP models.

This paper has performed a systematic literature review of 77 studies as per Kitchenham SLR guidelines. First, the primary studies are thoroughly examined and their quality is assessed. Secondly, the data is extracted which depicts the type of ML/DL methods used, and empirical validation for predicting vulnerable components (datasets, data balancing techniques, cross-validation methods, evaluation measures, HPO methods). Thirdly, the comparison among ML/DL techniques has been captured. The findings of this study are as follows:

- The current study collected 32 different ML and 5 DL techniques among 77 primary studies. DT, LR, NB, RF, and SVM are highly used. The tools for implementing these algorithms have also been mentioned.

- There exist five different feature types; metrics and text features are highly used.
- The study has also illustrated the data preprocessing methods which are feature extraction, feature reduction, and feature selection methods
- Thirty-seven different datasets are collected and 29 data sources are available. The datasets are mostly in PHP, C/C++, and Java programming languages. Other languages are Python and JavaScript.
- Twenty-five primary studies have used nine different data balancing techniques.
- Twenty-six different performance measures are collected from different studies. Accuracy, precision, recall, and F1-Score are used widely.
- Twenty-two studies have used HPO methods to increase the performance of SVP models
- Forty studies have shown the comparison among ML/DL techniques.

The following guidelines can be used for carrying out research in the future on SVP using ML/DL techniques

- More studies should incorporate data balancing techniques as vulnerability datasets are imbalanced as mentioned in this paper.
- Evolutionary algorithms can be used for feature selection to optimize SVP models
- The hyperparameters should be tuned to increase productivity. By far, only 22 studies have implemented HPO methods.

AUTHOR'S CONTRIBUTION

Deepali Bassi and Hardeep Singh contributed to the study's conception and design. Material preparation, data collection, and analysis were performed by Deepali Bassi. Hardeep Singh read and approved the final manuscript.

FUNDING

No funding was received to assist with the preparation of this manuscript.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest. We have no relevant financial or non-financial interests to disclose. We have no competing interests to declare that are relevant to the content of this article.

APPENDIX QUALITY ASSESSMENT SCORE

See Table 17.

REFERENCES

[1] S. Matteson. (2018). *Software Failure Caused $1.7 Trillion in Financial Losses in 2017*. [Online]. Available: https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/

[2] K. Bissell and RMLPD Cin. (2019). *Ninth Annual Cost of Cybercrime Study*. [Online]. Available: https://www.accenture.com/us-en/insights/security/cost-cybercrime-study

[3] NIOSA Technology. (2020). *National Vulnerability Database—Statistics*. [Online]. Available: https://nvd.nist.gov/vuln/search/statistics

[4] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Softw. Eng.*, vol. 18, pp. 25–59, Dec. 2013, doi: 10.1007/s10664-011-9190-8.

[5] H. Shahriar and M. Zulkernine, "Mitigating program security vulnerabilities: Approaches and challenges," *ACM Comput. Surveys*, vol. 44, no. 3, pp. 1–46, Jun. 2012, doi: 10.1145/2187671.2187673.

[6] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Comput. Surveys*, vol. 50, no. 4, pp. 1–36, Jul. 2018.

[7] Z. Li and Y. Shao, "A survey of feature selection for vulnerability prediction using feature-based machine learning," in *Proc. 11th Int. Conf. Mach. Learn. Comput.*, Feb. 2019, pp. 36–42, doi: 10.1145/3318299.3318345.

[8] H. Hanif, M. H. N. M. Nasir, M. F. A. Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *J. Netw. Comput. Appl.*, vol. 179, Apr. 2021, Art. no. 103009.

[9] R. Croft, Y. Xie, and M. A. Babar, "Data preparation for software vulnerability prediction: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 49, no. 3, pp. 1044–1063, Mar. 2023.

[10] B. Kitchenham, "Procedures for performing systematic reviews," Keele Univ., Keele, U.K., Tech. Rep. TR/SE-0401, vol. 33, no. 2004, 2004, pp. 1–26.

[11] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering—A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, Jan. 2009, doi: 10.1016/j.infsof.2008.09.009.

[12] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," Softw. Eng. Group, School Comput. Sci. Math., Keele Univ., Keele, U.K., Tech. Rep. EBSE-2007-01, 2007.

[13] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, Jan. 2012, doi: 10.1016/j.infsof.2011.09.002.

[14] W. Claes, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. ACM Int. Conf. Ser.*, 2014, pp. 1–10, doi: 10.1145/2601248.2601268.

[15] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proc. 14th ACM Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2007, pp. 529–540, doi: 10.1145/1315245.1315311.

[16] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: Learning to classify vulnerabilities and predict exploits," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*. New York, NY, USA: Association for Computing Machinery, 2010, pp. 105–114, doi: 10.1145/1835804.1835821.

[17] V. H. Nguyen and L. M. S. Trans, "Predicting vulnerable software components with dependency graphs," in *Proc. 6th Int. Workshop Secur. Meas. Metrics (MetriSec)*. New York, NY, USA: Association for Computing Machinery, 2010, pp. 1–8, doi: 10.1145/1853919.1853923.

[18] T. Zimmermann, C. Science, and N. Carolina, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *Proc. 3rd Int. Conf. Softw. Test., Verification Validation*, 2010, pp. 421–428, doi: 10.1109/ICST.2010.32.

[19] Y. Shin and L. Williams, "An initial study on the use of execution complexity metrics as indicators of software vulnerabilities," in *Proc. 7th Int. Workshop Softw. Eng. for Secure Syst.* New York, NY, USA: Association for Computing Machinery, May 2011, pp. 1–7, doi: 10.1145/1988630.1988632.

[20] B. Smith and L. Williams, "Using SQL hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities," in *Proc. 4th IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2011, pp. 220–229, doi: 10.1109/ICST.2011.15.

[21] L. K. Shar and H. B. K. Tan, "Predicting common web application vulnerabilities from input validation and sanitization code patterns," in *Proc. 27th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*. New York, NY, USA: Association for Computing Machinery, Jun. 2012, pp. 310–313, doi: 10.1145/2351676.2351733.

[22] L. K. Shar, H. B. K. Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, San Francisco, CA, USA, 2013, pp. 642–651, doi: 10.1109/ICSE.2013.6606610.

[23] A. A. Younis and Y. K. Malaiya, "Using software structure to predict vulnerability exploitation potential," in *Proc. IEEE 8th Int. Conf. Softw. Secur. Rel.-Companion*, San Francisco, CA, USA, Jun. 2014, pp. 13–18, doi: 10.1109/SERE-C.2014.17.

[24] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *Proc. Int. Symp. Softw. Reliab. Eng. (ISSRE)*, 2014, pp. 23–33, doi: 10.1109/ISSRE.2014.32.

[25] I. Abunadi and M. Alenezi, "Towards cross project vulnerability prediction in open source web applications," in *Proc. Int. Conf. Eng. MIS (ICEMIS)*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 1–5, doi: 10.1145/2832987.2833051.

[26] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 426–437, doi: 10.1145/2810103.2813604.

[27] L. K. Shar, L. C. Briand, and H. B. K. Tan, "web application vulnerability prediction using hybrid program analysis and machine learning," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 6, pp. 688–707, Nov. 2015, doi: 10.1109/TDSC.2014.2373377.

[28] M. K. Gupta, M. C. Govil, and G. Singh, "Predicting cross-site scripting (XSS) security vulnerabilities in web applications," in *Proc. 12th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Songkhla, Thailand, Jul. 2015, pp. 162–167, doi: 10.1109/JCSSE.2015.7219789.

[29] Y. Tang, F. Zhao, Y. Yang, H. Lu, Y. Zhou, and B. Xu, "Predicting vulnerable components via text mining or software metrics? An effort-aware perspective," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur.*, Vancouver, BC, Canada, Aug. 2015, pp. 27–36, doi: 10.1109/QRS.2015.15.

[30] B. M. Padmanabhuni and H. B. K. Tan, "Buffer overflow vulnerability prediction from x86 executables using static analysis and machine learning," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Taichung, Taiwan, vol. 2, Jul. 2015, pp. 450–459, doi: 10.1109/COMPSAC.2015.78.

[31] J. Stuckman, J. Walden, and R. Scandariato, "The effect of dimensionality reduction on software vulnerability prediction models," *IEEE Trans. Rel.*, vol. 66, no. 1, pp. 17–37, Mar. 2017, doi: 10.1109/TR.2016.2630503.

[32] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun, and S. Li, "Combining software metrics and text features for vulnerable file prediction," in *Proc. 20th Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, Gold Coast, QLD, Australia, 2015, pp. 40–49, doi: 10.1109/ICECCS.2015.15.

[33] A. Ibrahim and A. Mamdouh, "An empirical investigation of security vulnerabilities within web applications," *J. Universal Comput. Sci.*, vol. 22, pp. 537–551, Jun. 2016.

[34] M. K. Gupta, M. C. Govil, and G. Singh, "Text-mining based predictive model to detect XSS vulnerable files in web applications," in *Proc. Annu. IEEE India Conf. (INDICON)*, New Delhi, India, Dec. 2015, pp. 1–6, doi: 10.1109/INDICON.2015.7443332.

[35] S. Moshtari and A. Sami, "Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction," in *Proc. 31st Annu. ACM Symp. Appl. Comput.* New York, NY, USA: Association for Computing Machinery, Apr. 2016, pp. 1415–1421, doi: 10.1145/2851613.2851777.

[36] A. Hovsepyan, R. Scandariato, and W. Joosen, "Is newer always better: The case of vulnerability prediction models," in *Proc. 10th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.* New York, NY, USA: Association for Computing Machinery, Sep. 2016, pp. 1–6, doi: 10.1145/2961111.2962612.

[37] H. Alves, B. Fonseca, and N. Antunes, "Experimenting machine learning techniques to predict vulnerabilities," in *Proc. 7th Latin-Amer. Symp. Dependable Comput. (LADC)*, Cali, Colombia, 2016, pp. 151–156, doi: 10.1109/LADC.2016.32.

[38] Y. Pang, X. Xue, and A. S. Namin, "Early identification of vulnerable software components via ensemble learning," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Anaheim, CA, USA, Dec. 2016, pp. 476–481, doi: 10.1109/ICMLA.2016.0084.

[39] M. Davari and M. Zulkernine, "Analysing vulnerability reproducibility for Firefox browser," in *Proc. 14th Annu. Conf. Privacy, Secur. Trust (PST)*, Auckland, New Zealand, 2016, pp. 674–681, doi: 10.1109/PST.2016.7906955.

[40] S. E. Sahin and A. Tosun, "A conceptual replication on predicting the severity of software vulnerabilities," in *Proc. 23rd Int. Conf. Eval. Assessment Softw. Eng. (EASE)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 244–250, doi: 10.1145/3319008.3319033.

[41] S. Jha, R. Kumar, L. H. Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma, and H. V. Long, "Deep learning approach for software maintainability metrics prediction," *IEEE Access*, vol. 7, pp. 61840–61855, 2019, doi: 10.1109/ACCESS.2019.2913349.

[42] B. L. Bullough, A. K. Yanchenko, C. L. Smith, and J. R. Zipkin, "Predicting exploitation of disclosed software vulnerabilities using open-source data," in *Proc. 3rd ACM Int. Workshop Secur. Privacy Analytics*. New York, NY, USA: Association for Computing Machinery, Mar. 2017, pp. 45–53, doi: 10.1145/3041008.3041009.

[43] A. Rahman, P. Pradhan, A. Partho, and L. Williams, "Predicting Android application security and privacy risk with static code metrics," in *Proc. IEEE/ACM 4th Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft)*, May 2017, pp. 149–153, doi: 10.1109/MOBILESoft.2017.14.

[44] Y. Pang, X. Xue, and H. Wang, "Predicting vulnerable software components through deep neural network," in *Proc. Int. Conf. Deep Learn. Technol. (ICDLT)*. New York, NY, USA: Association for Computing Machinery, 2017, pp. 6–10, doi: 10.1145/3094243.3094245.

[45] H. Zhuobing, L. Xiaohong, X. Zhenchang, L. Hongtao, and F. Zhiyong, "Learning to predict severity of software vulnerability using only vulnerability description," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 125–136, doi: 10.1109/ICSME.2017.52.

[46] K. Z. Sultana and B. Williams, "Evaluating micro patterns and software metrics in vulnerability prediction," in *Proc. 6th Int. Workshop Softw. Mining (SoftwareMining)*, 2017, pp. 40–47, doi: 10.1109/SOFTWAREMINING.2017.8100852.

[47] K. Z. Sultana, "Towards a software vulnerability prediction model using traceable code patterns and software metrics," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Oct. 2017, pp. 1022–1025, doi: 10.1109/ASE.2017.8115724.

[48] A. Manar and T. Daniel, "When do changes induce software vulnerabilities?" in *Proc. IEEE 3rd Int. Conf. Collaboration Internet Comput. (CIC)*, 2017, pp. 59–66, doi: 10.1109/CIC.2017.00020.

[49] C. Catal, A. Akhan, E. Ecem, and A. Meltem, "Development of a software vulnerability prediction web service based on artificial neural networks," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*, 2017, pp. 59–67, doi: 10.1007/978-3-319-67274-8_6.

[50] S. M. Imtiaz and T. Bhowmik, "Towards data-driven vulnerability prediction for requirements," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 744–748, doi: 10.1145/3236024.3264836.

[51] S. Wei, H. Zhong, C. Shan, L. Ye, X. Du, and M. Guizani, "Vulnerability prediction based on weighted software network for secure software building," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Sep. 2018, pp. 1–6, doi: 10.1109/GLOCOM.2018.8647583.

[52] Q. Chen, L. Bao, L. Li, X. Xia, and L. Cai, "Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Nara, Japan, 2018, pp. 345–354, doi: 10.1109/APSEC.2018.00049.

[53] K. Z. Sultana, B. J. Williams, and A. Bosu, "A comparison of nanopatterns vs. software metrics in vulnerability prediction," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Nara, Japan, 2018, pp. 355–364, doi: 10.1109/APSEC.2018.00050.

[54] S. E. Sahin and A. Tosun, "A conceptual replication on predicting the severity of software vulnerabilities," in *Proc. Eval. Assessment Softw. Eng. (EASE)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 244–250, doi: 10.1145/3319008.3319033.

[55] P. K. Kudjo and J. Chen, "A cost-effective strategy for software vulnerability prediction based on bellwether analysis," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal. (ISSTA)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 424–427, doi: 10.1145/3293882.3338985.

[56] M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. L. Traon, and M. Harman, "The importance of accounting for real-world labelling when predicting software vulnerabilities," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 695–705, doi: 10.1145/3338906.3338941.

[57] K. Zhang, "A machine learning based approach to identify SQL injection vulnerabilities," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA, 2019, pp. 1286–1288, doi: 10.1109/ASE.2019.00164.

[58] R. Ferenc, P. Hegedüs, P. Gyimesi, G. Antal, D. Bán, and T. Gyimóthy, "Challenging machine learning algorithms in predicting vulnerable JavaScript functions," in *Proc. IEEE/ACM 7th Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng. (RAISE)*, Montreal, QC, Canada, May 2019, pp. 8–14, doi: 10.1109/RAISE.2019.00010.

[59] T. -Y. Chong, V. Anu, and K. Z. Sultana, "Using software metrics for predicting vulnerable code-components: A study on Java and Python open source projects," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE) IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC)*, New York, NY, USA, Jun. 2019, pp. 98–103, doi: 10.1109/CSE/EUC.2019.00028.

[60] A. Kaya, A. S. Keceli, C. Catal, and B. Tekinerdogan, "The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models," *J. Softw., Evol. Process*, vol. 31, no. 9, pp. 1–25, Sep. 2019, doi: 10.1002/smr.2164.

[61] P. K. Kudjo, J. Chen, S. Mensah, and A. Richard, "Predicting vulnerable software components via bellwethers," in *Proc. 12th Chin. Conf. (CTCIS)*, Wuhan, China, Oct. 2019, pp. 389–407, doi: 10.1007/978-981-13-5913-2_24.

[62] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, and Z. Wang, "Large-scale empirical studies on effort-aware security vulnerability prediction methods," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 70–87, Mar. 2020, doi: 10.1109/TR.2019.2924932.

[63] Y. Movahedi, M. Cukier, and I. Gashi, "Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models," *Comput. Secur.*, vol. 87, Nov. 2019, Art. no. 101596, doi: 10.1016/j.cose.2019.101596.

[64] R. Shu, T. Xia, L. Williams, Laurie, and T. Menzies, "Better security bug report classification via hyperparameter optimization," 2019, *arXiv:1905.06872*.

[65] P. K. Kudjo, S. B. Aformaley, S. Mensah, and J. Chen, "The significant effect of parameter tuning on software vulnerability prediction models," in *Proc. Companion 19th IEEE Int. Conf. Softw. Quality Reliab. Secur. (QRS-C)*, Jun. 2019, pp. 526–527, doi: 10.1109/QRS-C.2019.00107.

[66] N. Muhammad, F. Humera, I. Muhammad, A. Muhammad, and R. Kamran, "Predicting web vulnerabilities in web applications based on machine learning," in *Proc. 1st Int. Conf. (INTAP)*, Bahawalpur, Pakistan, Oct. 2019, pp. 473–484, doi: 10.1007/978-981-13-6052-7_41.

[67] P. K. Kudjo, J. Chen, S. Mensah, R. Amankwah, and C. Kudjo, "The effect of bellwether analysis on software vulnerability severity prediction models," *Softw. Quality J.*, vol. 28, no. 4, pp. 1413–1446, Dec. 2020, doi: 10.1007/s11219-019-09490-1.

[68] N. Bhatt, A. Anand, and V. S. S. Yadavalli, "Exploitability prediction of software vulnerabilities," *Qual. Rel. Eng. Int.*, vol. 37, no. 2, pp. 648–663, Mar. 2021, doi: 10.1002/qre.2754.

[69] K. Z. Sultana, V. Anu, and T. Chong, "Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach," *J. Softw., Evol. Process*, vol. 33, no. 3, p. 2303, Mar. 2021, doi: 10.1002/smr.2303.

[70] Z. Yuyue, L. Yangyang, Y. Tengfei, and X. Haiyong, "Suzzer: A vulnerability-guided fuzzer based on deep learning," in *Proc. Int. Conf. Inf. Secur. Cryptol.*, 2020, pp. 134–153, doi: 10.1007/978-3-030-42921-8_8.

[71] F. Alenez and C. Tsokos, "Machine learning approach to predict computer operating systems vulnerabilities," in *Proc. 3rd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS)*, 2020, pp. 1–6, doi: 10.1109/ICCAIS48893.2020.9096731.

[72] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak, and L. Karaçay, "Vulnerability prediction from source code using machine learning," *IEEE Access*, vol. 8, pp. 150672–150684, 2020, doi: 10.1109/ACCESS.2020.3016774.

[73] X. Chen, Z. Yuan, Z. Cui, D. Zhang, and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," *IET Softw.*, vol. 15, no. 1, pp. 75–89, Feb. 2021.

[74] T. Aladics, J. Jász, and R. Ferenc, "Bug prediction using source code embedding based on Doc2Vec," in *Proc. Int. Conf. Comput. Sci. Appl.*, 2021, pp. 382–397, doi: 10.1007/978-3-030-87007-2_27.

[75] C. B. Şahin, Ö. B. Dinler, and L. Abualigah, "Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features," *Appl. Intell.*, vol. 51, pp. 8271–8287, Mar. 2021, doi: 10.1007/s10489-021-02324-3.

[76] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for predicting vulnerable software components," *IEEE Trans. Softw. Eng.*, vol. 47, no. 1, pp. 67–85, Jan. 2021, doi: 10.1109/TSE.2018.2881961.

[77] C. B. Şahin, "The role of vulnerable software metrics on software maintainability prediction," *Eur. J. Sci. Technol.*, vol. 23, no. 3, pp. 686–696, Apr. 2021.

[78] A. Bagheri and P. Hegedűs, "A comparison of different source code representation methods for vulnerability prediction in Python," in *Proc. Int. Conf. Quality Inf. Commun. Technol.*, 2021, pp. 267–281.

[79] K. Filus, P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe, "Efficient feature selection for static analysis vulnerability prediction," *Sensors*, vol. 21, no. 4, p. 1133, Feb. 2021, doi: 10.3390/s21041133.

[80] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, "How to better distinguish security bug reports (using dual hyperparameter optimization)," *Empirical Softw. Eng.*, vol. 26, no. 3, May 2021, doi: 10.1007/s10664-020-09906-8.

[81] T. Viszkok, P. Hegedűs, and R. Ferenc, "Improving vulnerability prediction of Javascript functions using process metrics," in *Proc. 16th Int. Conf. Softw. Technol.*, 2021, pp. 185–195, doi: 10.5220/0010558501850195.

[82] Y. Li, S. Ji, C. Lyu, Y. Chen, J. Chen, Q. Gu, C. Wu, and R. Beyah, "V-fuzz: Vulnerability prediction-assisted evolutionary fuzzing for binary programs," *IEEE Trans. Cybern.*, vol. 52, no. 5, pp. 3745–3756, May 2022, doi: 10.1109/TCYB.2020.3013675.

[83] I. Kalouptsoglou, M. Siavvas, D. Kehagias, A. Chatzigeorgiou, and A. Ampatzoglou, "An empirical evaluation of the usefulness of word embedding techniques in deep learning-based vulnerability prediction," in *Security in Computer and Information Sciences*, vol. 1596, E. Gelenbe, M. Jankovic, D. Kehagias, A. Marton, and A. Vilmos, Eds. Cham, Switzerland: Springer, 2022, doi: 10.1007/978-3-031-09357-9_3.

[84] G. Jabeen, S. Rahim, and W. Afzal, "Machine learning techniques for software vulnerability prediction: A comparative study," *Appl. Intell.*, vol. 52, pp. 17614–17635, Apr. 2022, doi: 10.1007/s10489-022-03350-5.

[85] M. Fu and C. Tantithamthavorn, "LineVul: A transformer-based line-level vulnerability prediction," in *Proc. IEEE/ACM 19th Int. Conf. Mining Softw. Repositories (MSR)*, Pittsburgh, PA, USA, May 2022, pp. 608–620, doi: 10.1145/3524842.3528452.

[86] R. Shu, T. Xia, L. Williams, and T. Menzies, "Dazzle: Using optimized generative adversarial networks to address security data class imbalance issue," in *Proc. 19th Int. Conf. Mining Softw. Repositories (MSR)*. New York, NY, USA: Association for Computing Machinery, 2022, pp. 144–155, doi: 10.1145/3524842.3528437.

[87] I. Kalouptsoglou, M. Siavvas, D. Kehagias, A. Chatzigeorgiou, and A. Ampatzoglou, "Examining the capacity of text mining and software metrics in vulnerability prediction," *Entropy*, vol. 24, p. 651, Mar. 2022, doi: 10.3390/e24050651.

[88] W. Rhmann, "Software vulnerability prediction using grey wolf-optimized random forest on the unbalanced data sets," *Int. J. Appl. metaheuristic Comput.*, vol. 13, no. 1, pp. 1–15, Jan. 2022, doi: 10.4018/IJAMC.292508.

[89] A. W. Marashdih, Z. F. Zaaba, and K. Suwais, "Predicting input validation vulnerabilities based on minimal SSA features and machine learning," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 10, pp. 9311–9331, Nov. 2022, doi: 10.1016/j.jksuci.2022.09.010.

[90] S. Ganesh, F. Palma, and T. Olsson, "Are source code metrics 'good enough' in predicting security vulnerabilities?" *Data*, vol. 7, p. 127, Aug. 2022, doi: 10.3390/data7090127.

[91] D. Bassi and H. Singh, "Optimizing hyperparameters for improvement in software vulnerability prediction models," in *Proc. Adv. Distrib. Comput. Mach. Learn. (ICADCML)*. Singapore: Springer Nature, 2022, pp. 533–544.

DEEPALI BASSI is currently pursuing the Ph.D. degree with the Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab, India. Her research interests include software engineering and machine learning.

HARDEEP SINGH is currently a Professor with the Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab, India. His research interests include software engineering and information systems.