IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 May 2022, accepted 11 July 2022, date of publication 15 July 2022, date of current version 22 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3191115

METHODS

Developer's Roadmap to Design Software Vulnerability Detection Model Using Different AI Approaches

POOJA S[1], CHANDRAKALA C. B.[1], AND LAIJU K. RAJU[2]

[1]Department of Information & Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

[2]dltledgers India Private Limited, Thiruvananthapuram, Kerala 695035, India

Corresponding author: Chandrakala C. B. (chandrakala.cb@manipal.edu)

ABSTRACT Automatic software vulnerability detection has caught the eyes of researchers as because software vulnerabilities are exploited vehemently causing major cyber-attacks. Thus, designing a vulnerability detector is an inevitable approach to eliminate vulnerabilities. With the advances of Natural language processing in the application of interpreting source code as text, AI approaches based on Machine Learning, Deep Learning and Graph Neural Network have impactful research works. The key requirement for developing an AI based vulnerability detector model from a developer perspective is to identify which AI model to adopt, availability of labelled dataset, how to represent essential feature and tokenizing the extracted feature vectors, specification of vulnerability coverage with detection granularity. Most of the literature review work explores AI approaches based on either Machine Learning or Deep Learning model. The existing literature work either highlight only feature representation technique or identifying granularity level and dataset. A qualitative comparative analysis on ML, DL, GNN based model is presented in this work to get a complete picture on VDM thus addressing the challenges of a researcher to choose suitable architecture, feature representation and processing required for designing a VDM. This work focuses on putting together all the essential bits required for designing an automated software vulnerability detection model using any various AI approaches.

INDEX TERMS Machine learning, deep learning, graph neural network, feature representation, tokenization, granularity.

I. INTRODUCTION

Aflaw in software code allows access to internal system or network and is termed as cyber security vulnerability. These vulnerabilities if not patched, leave the system open for hacking, account transfers, malware attacks etc. The National Vulnerability Database (NVD) [1] is fed by Common vulnerabilities and Exposure (CVE) list [2] has currently over 1,50,000 entries [3]. In 2021 the NVD holds 21,957 vulnerabilities which is much in higher compared to 18,362 in 2020, 17,382 in 2019. According to Edgescan's report [4], organizations with more than 101-1000 employees have largest portion of high-risk vulnerabilities. Companies with 10,000+ employees see largest portion of medium and critical-risk vulnerabilities.

Now software vulnerability detection can be modeled as Natural Language Processing (NLP) problem with source code treated as texts. Thus, recent advances in Artificial Intelligence (AI) models like machine learning (ML) [5] models, deep learning (DL) [6] models, Graph Neural Network (GNN) [7], tries to addresses software vulnerability detection. ML models is successful at object detection, speech recognition and with deep learning paradigm it is capable in capturing hidden patterns of videos, images, text. GNN is a class of deep learning methods [8] which infers data described using graphs. GNN has seen an upcoming potential in its use case for software vulnerability detection [9]. The vulnerability detection models aim for binary classification

The associate editor coordinating the review of this manuscript and approving it for publication was Daniel Augusto Ribeiro Chaves.

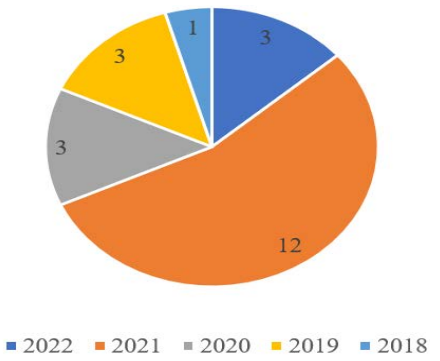

FIGURE 1. Spread of literature from year 2018 to 2022 for vulnerability detection in C/C++ and java source code.

i.e. categorizing input code as vulnerable or secure code; or as a multiclass classification, additionally classifying into particular type of vulnerability.

This systematic literature review aims to analyze the different AI approaches used to analyze software vulnerabilities for C/C++ and java programming language. The high-level languages like java and C are designed for better human interaction and expression. Thus, have a likeness with Natural Language Processing (NLP). Vulnerability detection is a borrowed concept from NLP. This work will act as a guide for future researches in considering various aspects while designing vulnerability detection model for different AI approaches. The key aspects highlighted in this study are dataset availability, targeted vulnerabilities, feature representation, feature vectorization, algorithms used for training with respect to each of the AI approaches considered here. Thus, work gives a complete overview on the current trends of AI approaches used for software vulnerability detection model. The literature considered for this study ranges from the year 2018 to 2022 so that recent trends can be studied and evaluated. It is shown using Figure 1. The study has considered 3 papers for ML based approach, 9 papers for DL based approach and 10 papers for GNN based approach which targets vulnerability detection using C/ C++ and java based source code projects. This spread of paper is also represented in Figure 2.

The existing literature work [5], [10], [11], [12], [13] gives thorough review in the VDM based on either on Machine learning only or on Deep Learning model. Table 1. Presents the major contribution of the proposed work in comparison with other literature survey. [5] covers VDM based on ML, DL architecture. The work reviews feature representation used, vulnerability coverage considered, metrics used for performance evaluation and whether the VDM considered for review provides a binary /multiclass classifier. It lacks specification on granularity level archived by the literature work under consideration. Also, the paper has not considered the GNN based VDM which is a research area in its infancy. [10] reviews vulnerability detection model based on DL /neural network based models. Work has highlighted feature representation techniques, granularity, dataset labelling. It has not listed the metrics used to measure the performance of the VDM. The work also does not detail the key aspects for ML based VDM. [11] reviewed articles only based on DL models. The work has classified literature work based on feature representation based semantic source code representation, code gadget representation and binary representation of source code. It lacks details about vectorization technique used for VDM, has not mentioned metrics and type of classifier the reviewed work could achieve. [12] reviews work on android based malware detection using ML and DL based model. The literature work has categorized based on different DL architectures available. The work has discussed about feature representation, feature vectorization techniques, metrics used for evaluating performance. But GNN based work is not highlighted in the work. [13] focuses only on ML models and specify different analysis technique of source code based on static analysis, symbolic execution and fuzzing. Less focus is given in detailing the vulnerability granularity detection, dataset availability, vulnerability coverage. The proposed work thus collects the key aspects for VDM based on ML, DL and GNN based architecture. It presents a clear picture on the methods used for feature representation, feature vectorization, granularity level achieved by the work under consideration, availability of labelled dataset, metrics considered for evaluation, and whether the classifier is binary or multiclass.

Section 2 gives a brief overview on the key aspects required for a software vulnerability detection model. Section 3 gives a detailed report based on literature review done on each of the aspects. Section 5 gives the conclusion of the study.

II. KEY ASPECTS CONSIDERED FOR VULNERABILITY DETECTION MODEL (VDM)

Software vulnerability detection model based on AI algorithm, requires availability of labelled dataset for the targeted programming languages. The researcher cannot feed the source code or intermediate code directly to the classifier. It needs to represented in machine understandable form using feature representation and feature vectorization technique. Based on the availability of labelled vulnerability classes, the model can be trained to output as a multiclass or a binary classifier specifying the granularity of detection. The researcher should ensure availability of standard dataset. These key aspects is shown in Figure 3. Dataset contains vulnerabilities is identified by using standard term, common vulnerabilities and exposure identifier (CVE-ID) or common weakness enumeration identifier (CWE-ID). For the current study, Java and C/C++ open source datasets are considered and reviewed. The other paramount importance is given to the ground truth labelling of the dataset using open source tools like PMD or availability of labelled dataset itself. Labelled dataset is based on the type of vulnerabilities the source code has. Vulnerabilities targeted for C/C++ source code are mostly based on buffer overflow vulnerability identified by (CWE-119), resource management error (CWE-399)

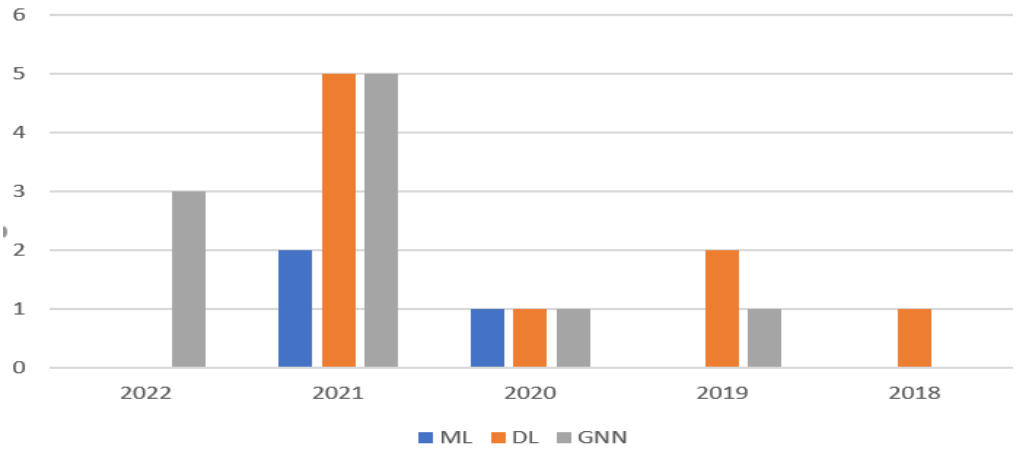

FIGURE 2. Spread of literature on different AI based VDM from 2018 to 2022.

and improper resource lifetime control (CWE-644). For java source code, most common vulnerabilities considered are SQL injections, denial of service, cross-site scripting etc. These vulnerabilities are identified by CWE-ID's, CWE-89, CWE-119, CWE-120, CWE-80.

The next stage of VDM is the representation of feature. Feature representation can be categorized as text-based, sequence-based or graph-based. Text-based feature representation converts source code into textual tokens. These tokens are software metrics [14] extracted from program source code such as cyclomatic complexity, average number of lines, return type of functions, parameters, number of statements, number of local methods, number of blank lines, maximum depth of class in inheritance tree. Sequence based feature representation comprises sequence of code based on syntactic similarity, function call sequences. Code gadget can accurately capture syntax or semantics of the vulnerability candidate obtained from the dataset. It can be extracted using commercial tools only, Checkmarx [15]. The other open source tools like Flawfinder and RATS have simple parsers and imperfect rules Graph based feature representations makes use of graph data model [16] which is a fairly established field in program analysis. Abstract Syntax Tree (AST) [17] helps in understanding fundamental structure of program which in turn helps in identifying syntactic errors. It is an ordered structured representation of source code. It presents code, content, context and control flow of the source code. Control Flow Graph (CFG) [18] describes all the path traversed when program is executed. CFG nodes indicate statement and edges indicate transfer of control. It has two variants inter-procedural and intra-procedural. Data Flow Graph (DFG) [18] used to track usage of variables through CFG. DFG edge presents subsequent access or manipulation of same variables. Data Dependence Graph (DDG) [18] is multi-digraph where vertices are program statements and edges are flow from source statement to destination statement. It requires both intra-procedural control flow information and inter-procedural control flow. Control Dependency Graph (CDG) [18] is a directed tree where vertices are program statements and each vertex have control dependency on parent vertex. It is generated by analyzing AST. If there is a relationship between nodes Sa and Sb, i.e. Sa-> Sb. By analyzing node Sa, it can be analyzed if node Sb will be executed. Program Dependency Graph (PDG) [18] consists of two subtypes of graph mainly control dependence graph (CDG) and data dependence graph (DDG). Code Property Graph (CPG) [18] is a combination of AST, CFG, PDG. The different types of feature representation is shown in Figure 4. AI models cannot interpret text as it is, in order for the text to be machine readable, features need to be vectorized using feature vectorization technique. The source code or intermediate code to be fed to the AI based model need to be vectorized using technique based on syntactic representation and semantic representation. Syntactic word representation does not capture words like ''airplane'', ''aeroplane'', ''plane'' and ''aircraft'' are often used in same context. Few types which are the current trends are N gram, Bag-of-words, TF-IDF, Word2Vec, etc. In N-Gram [19] set of n-word which occur in that order in the dictionary of words. It is common to use 1-gram, 2-gram, and 3-gram vector representation. The Bag-of-words (BoW) model [20] in turn is called as 1-gram representation. But it loses the order information of the words. It is used in NLP, document classification and retrieval in Machine learning etc. In Term Frequency-Inverse Document Frequency (TF-IDF) [21] it tells the frequency of the word in the dictionary. Inverse document frequency allots higher weights to word with higher or lower frequency term.

Word embedding is mapping of each word or phrase from dictionary to vector of n dimension. Mainly three different vectorization techniques are discussed here. Most of the literature work reviewed has used word2vec model. Other reviewed vectorization technique in this study is shown in Figure 5.

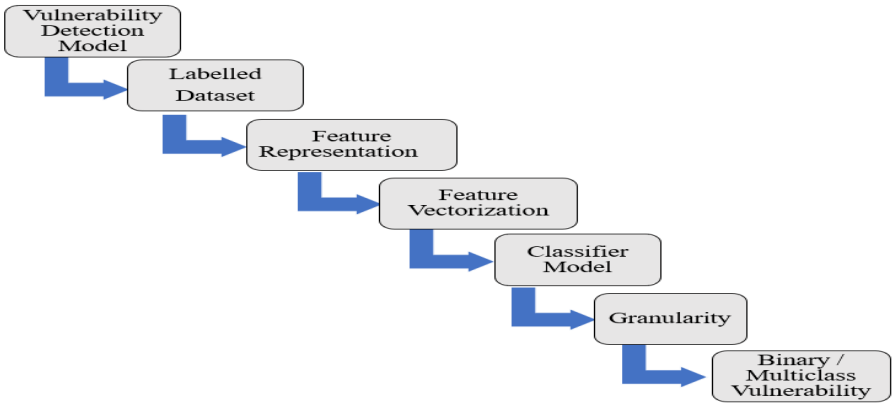

FIGURE 3. Key aspects of vulnerability detection model.

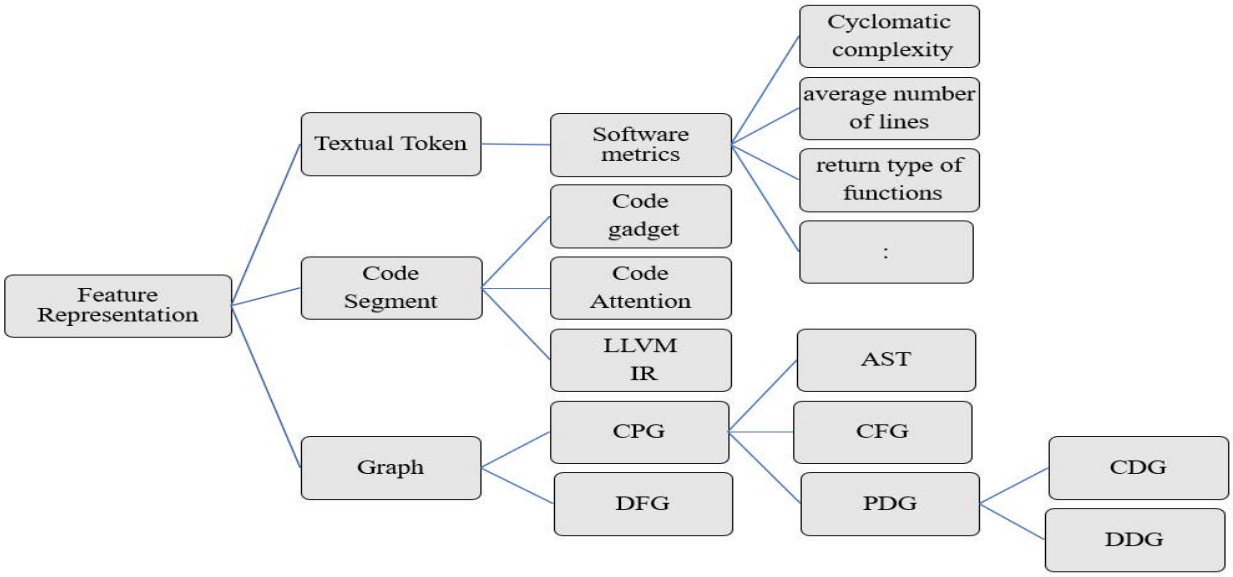

FIGURE 4. Different feature representation techniques used in AI based vulnerability detection model.

In addition to it the paper will discuss Index-based vectorization, Count vectorizer. The Index- based vectorization is the simplest method, wherein each word takes each number. Thus, different words take different number. So, codes which contain multiple words contain vectors of numbers, which then has to limited or padded to some threshold value, because classification model takes fixed size vectors. Count Vectorizer counts the frequency of the word in the entire training dataset. It is similar to bag-of-words. Vector length here will be total number of words in the dictionary. But it loses the order of word information. Word2Vec [22], [23] uses shallow neural network with hidden layers such as continuous bag-of words (CBOW) [24] and Skip-gram [19] model to create high dimension vector for each word. CBOW uses multiple word for a given target of words. Continuous skip-gram model predicts current word based on context. CBOW and continuous skip-gram keep syntactic and semantic information of sentences. In FastText [25] each word is represented using bag of character n-gram. Example for the given "word" and n = 3. Tri-grams may represent wo, or, rd, wor, ord. It uses Skip-gram model with default parameters. In Global Vectors for Word Representation (GloVe) [26] each word is represented by high dimension vector and trained based on nearby word over huge dictionary. Pre-trained word vectorizations can have 100, 200 or even 300 dimensions.

Once the vectorized tokens are extracted it is fed into the algorithms based on AI approaches. AI Models can detect vulnerabilities at varying granularity levels. Granularity of code varies from coarse granularity to finer granularity is shown in Figure 1. Coarse granularity again needs dependency on human expertise for pinpointing vulnerability location. It can vary from release level [27]–[29], file level [30], package level [14] to program level [31], [32]. Finer granularity can pinpoint vulnerability varying level ranging from

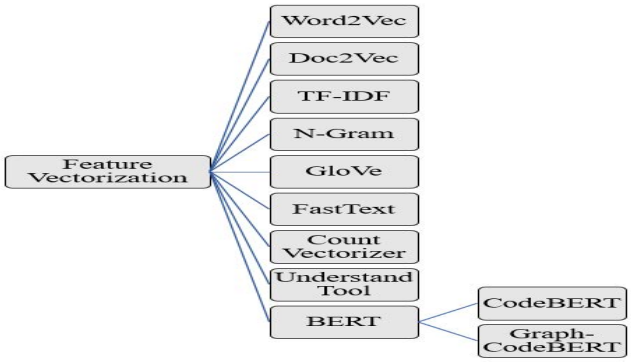

FIGURE 5. Different feature vectorization techniques used in AI based vulnerability detection model.

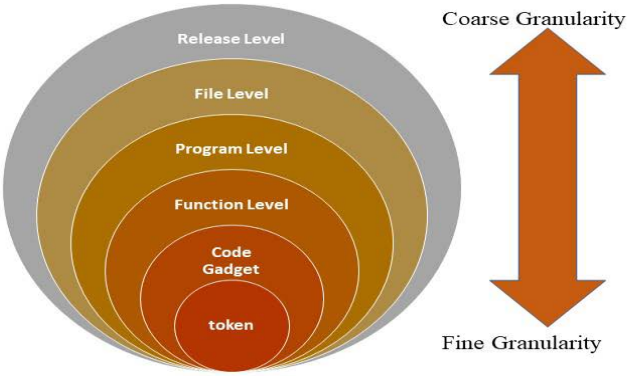

FIGURE 6. Varying granularity levels during vulnerability detection of source code analysis.

function level, to code fragment [10,28,33], to statement level and to the finest granularity, to token level [33], [34], [35]. Figure 6 represents the granularity for program source code analysis which can be achieved by vulnerability detection model based on conventional machine learning algorithm or deep learning algorithm or graph neural network algorithm.

Metrics used for Evaluation [36] considered for evaluating the proposed work are Precision, accuracy, recall, ROC curve and Kappa Statistics. The system should aim for low False Positive Rate and False Negative Rate and high Precision(P), Recall(R) and F1 measure(F1).

1) True Positive (TP) Rate [36] ratio of actual positives, which are predicted as positives. It is given using equation (1) [36], where tp is true positive and fn is false negative. Former in vulnerability context indicates detecting vulnerability when vulnerability exist in the system and latter indicates it does not detect condition when vulnerability exist in the system.

$$TPRate = \frac{tp}{tp + fn} \tag{1}$$

2) False Positive (FP) Rate [36] ratio of negative instance which are incorrectly labelled as positive. In vulnerabilities context, it can be termed as non-vulnerable instances which are termed as vulnerable. It is represented using equation (2) [36]

$$FPRate = \frac{fp}{tp + fp} \tag{2}$$

3) Precision [36] is the positively predicted instances which are actually classified as positive. It can be given using the equation (3) [36]

$$P = \frac{tp}{tp + fp} \tag{3}$$

4) Recall [36] is the ratio of predicted vulnerabilities that are actually vulnerable to the total number of vulnerable classes in the system. It is given using equation (4) [36]. Higher Recall means a smaller number of vulnerabilities which goes undetected. Recall is directly related to increase in FP rate.

$$R = \frac{tp}{tp + fn} \tag{4}$$

5) F-Measure [37] is weighted average of precision and recall. It gives equal weightage to both precision and recall using harmonic mean. The formula for F-Measure is given in equation (6) [37] as follows:

$$F_\beta = \frac{(\beta^2 + 1)Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{5}$$

where β balances between Precision and Recall. If $\beta = 1$, indicates F1 measures equivalent to harmonic mean of precision and recall. If $\beta > 1$ it is more oriented towards precision else it is favoring recall.

6) kappa statistics relates observed accuracy with expected accuracy. It helps in evaluation classifiers among themselves. It is presented using equation (6), where Pr(a) and Pr(e) is observed agreement among experiments and latter is hypothetical probability of the raters.

$$k = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \tag{6}$$

7) Receiver Operating Characteristics (ROC) [36] Curve it analyzes the performance of the system in terms of TP rate and FP rate. It is used for organizing, visualizing and measure performance.

8) Intersection over Union (IoU) [38] is to evaluate the location precision of vulnerability detector. It is represented by equation (7) [38], where U indicate set of truly vulnerable code and V is detected set of vulnerable lines.

$$IoU = \frac{U \cap V}{U \cup V} \tag{7}$$

9) Matthews Correlation Coefficient (MCC) [39] predicts the degree to which the predicted model matches with

the ground truth labels. It is represented using equation (9) [39].

$$temp = (TP + FP)(TP + FN)(TN + FP)(TN + FN) \tag{8}$$

$$MCC = \frac{(TP \times TN - FP \times FN)}{\sqrt{(temp)}} \tag{9}$$

III. REVIEW OF THE KEY ASPECTS OF VDM BASED ON DIFFERENT AI APPROACHES

A. KEY ASPECT OF VDM ON MACHINE LEARNING MODEL

Key aspects such as feature representation, feature vectorization, granularity level, metrics used for performance evaluation is elaborated in this section. For machine learning based model, metrics or textual based input is used as feature representation. Fig 7 represents techniques used in keys aspects of an automated vulnerability detection model system based on ML model. Software metrics, such as class level and method level metrics are derived using Understand Tool [40] which is used in research work [14], [41]. This tool in addition to having the set of standard metrics also can perform vectorization of this metrics presented in the tool. Thus, it a compact tool which present metrics and generate vector for the corresponding metrics. The only drawback is this tool is it can be used for java source code only. [34] uses deep learning for representation of feature and feature vectorization. The technique used is CBOW [23] for feature representation and LSTM for feature vectorization. Thus, such type of feature representation and vectorization is able to achieve more inter-sequence information which Understand tool lacks. For collecting feature metrics, Understand tool is used for both [41] and [14]. Metrics at class, function, project, package level can be extracted. After the feature metrics is extracted, [14] used manual labelling which may have caused the work to have less accuracy, since manual labelling requires domain experts. [41] used already labelled instances from the Apache Release dataset [42]. Stanford SecuriBench dataset [43] used ESVD tool [44] to identify vulnerable classes and methods. Autor [41] performed Mann-Whitney U test [36] to identify significant set of metrics. Hence the accuracy of prediction is better than [14]. The ML model is able to achieve only a binary classifier but with finer granularity if combined with deep representation learning [34]. The limitation of [34] is that, this deep learning representation is used to detect only two types of vulnerabilities based on C/ C++ program code which are buffer overflow and resource management error. Another advantage of [34] over the other two current trends is that the [34] has used ensemble classifiers instead of experimenting with different conventional ML algorithms. [34] uses Logistic Regression (LR) and MultinomialNB(NB) as base classifiers and Random Forest as the final classifier. It can also be observed that granularity level achieved by [34] is word level, compared to class level and method level obtained in [41] and [14] respectively. ML model could only achieve to produce a binary classifier, which classifies the code as secure or vulnerable. Thus, further effort is required to achieve a multi-class classifier since that can effectively reduce the effort of developer.

1) REVIEW OF RELATED WORKS

The detailed review of software vulnerability detection model on machine learning is discussed and it is presented in Table 7. [41] aims to predict vulnerability for java projects at two levels of granularity, class level and function level using supervised ML algorithm like Support Vector Machine (SVM) and Logistic Regression (LR). The dataset consists of ApacheTomcat [42] releases 6 and 7, Apcahe CXF [45] and Stanford SecuriBench dataset [43]. The apache dataset can be found in [46]. The work predicts that class level metrics is able to classify dataset with better precision in dataset [45] and [43]. The dataset was not balanced hence a classBalancer filter [47] in WEKA 3.8 is applied which reweighs the instances in the data. The metrics for class level and function is extracted using commercial tool called Understand 4.0 [40]. [41] makes use of software metrics, Avg-Cyclomatic [48] which averages the McCabe's cyclomatic [48] complexity metric for class level and for method-level software metric which include structural complexity of functions, dependency on other method, parameters, return types it uses Max-Nesting [49]. The work is binary classifier. And is evaluated using metrics like precision, recall, accuracy.

[14] uses 32 supervised machine learning algorithm and focusses mainly on 3 types of vulnerabilities. The work is validated using tenfold cross validation and other statistical parameters like ROC curve, Kappa statistics [36], Recall, Precision. The feature metrics are selected at project, package, method and class level. It makes use of all the object-oriented metrics obtained from Understand Tool [40] developed by Scitools. Some of the metrics used are Avg-LineCode, CountClassBase, CountCoupleCoupled, Count-ClassDerived, CountLineCode and so on. Out of 32 algorithm J48, AdaBoostM1 and Local weighted Learning (LWL) are the best performers. J48 algorithm implements C4.5 decision tree. The dataset used is from previous work [50] and is balanced using stratified sampling.

[34] proposes for an automated vulnerability detection model using deep representation learning and ensemble classifiers [51]. Source code is tokenized into code sequence which consist token, its preceeding token and succeeding token. This token is vectorized using non- static embedding, namely continuous bag-of-words (CBOW) [23]. The word vectors are fed into concatenated Convolutional Neural Network (CNN) [52] and LSTM. The features created will learn more structure and semantics of data. These features are trained using ensemble classifiers [51] which are stacked in two stages. Stage 1 contain Logistic Regression (LR) and Multinomial Naïve Bayesian (MNB). This stack 1 output is fed as input to stack 2 which contain Random Forest Classifier. The work has tried to achieve low false positive and whilst maintaining a high recall rate. Dataset used for the proposed work includes from [1], [53], [54] for training, testing and validation. Also, the model is a binary classifier.

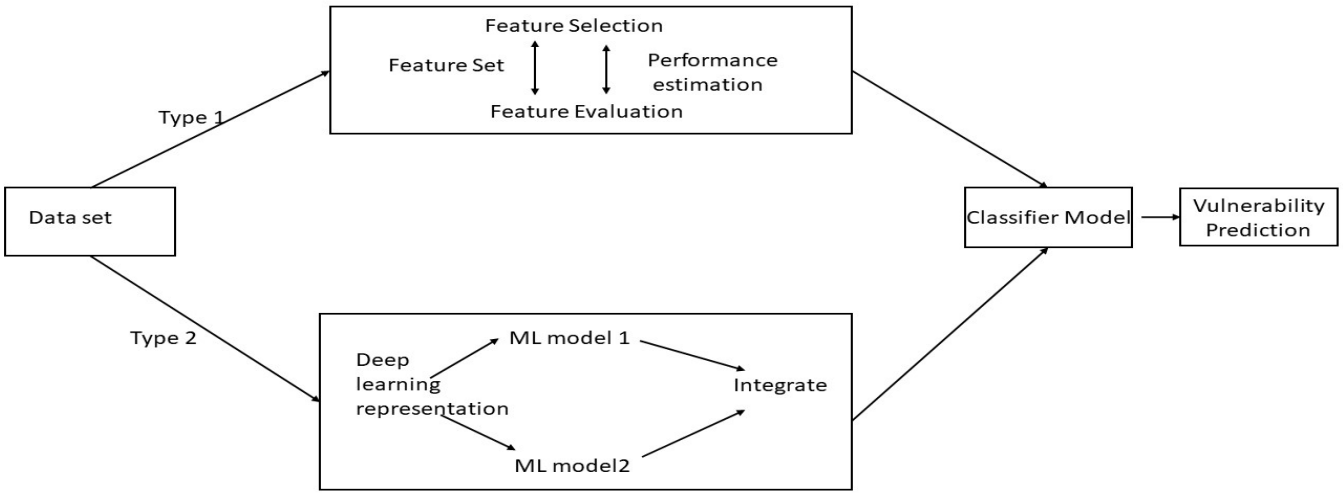

FIGURE 7. Represents specification of the key aspect of VDM based on ML model.

2) DISCUSSIONS

The most performing ML based VDM employs J48 algorithm, with class level metrics vectorized using Understand Tool. Though it is said that ensemble classifiers perform better than conventional ML algorithm, it is not so, as is proved by [14]. But it is also to be noted that, when using conventional algorithm, using feature representation and vectorization using word2vec+ CNN based representation learning helps the conventional ML algorithm to perform better as shown in the work [34]. Least performing ML algorithm is LR based VDM model which takes only input from Word2Vec. Also, the metrics generated from Understand tool performs better than Word2vec for conventional ML algorithm. This comparison is shown in Figure 8 which shows comparison of performance based on F1 and Recall score from the literature's considered for this work shown via Table 2.

B. KEY ASPECTS OF VDM ON DEEP LEARNING MODEL

Main difference between machine learning and deep learning is, former is probabilistic and latter is deterministic. Existing solutions rely on human experts for feature extraction and selection. DL relives humans of such a tedious task which is error prone, time consuming and require efficient human expertise. Deep learning is popular in vulnerability detection since it alleviates from manual feature detection and extraction process. Deep learning is successful in image processing and natural language processing using models on Recurrent Neural Network (RNNs) [55], Convolutional Neural Network (CNNs) [52] and Deep Belief Network(DBN) [56]. Feature representation used for DL models is mostly sequence based representation. Sequence based feature representation makes use of features such as sequence of statements, function call execution etc. [57] makes use of code gadget along with employing and comparing different vectorization method such as index-based, count vectorizer, and word2vec. Code gadget is generated from data flow-based analysis of code. [58] uses refined technique of code gadget for feature representation i.e. code attention which captures data dependence and control dependence relation. [33], [59] makes use of syntax based vulnerability candidate (SyVC) and sematics based vulnerability candidate(SeVC). SyVC is piece of code that bears some vulnerability generated by AST of program source code. SeVC is generated from intermediate code based on SyVC. SeVC capture semantic information induced from data dependency and control dependency graph. The DL models incorporates parser like clang [60] and joern's open source tool [16] to extract the AST or CFG from C/ C++ programs. These graph representation of program is vectorized using Word2vec tool. The vectorized token is then used to train DL model based on RNN and CNN. Usually it is observed that variants of RNN like BLSTM and BRNN is mostly used in existing literature work. The advantage of DL is higher accuracy, finer granularity [33], [58] and ability to map multiclass vulnerability [33], [58] which ML and GNN based vulnerability detector system fails to achieve. The only limitation is that the work on DL based VDM on java projects is not much explored. It can be attributed to absence of well labelled java open source projects with real projects. Figure 9 represents an automated vulnerability detection system based on DL model with the techniques used for each key aspect.

1) REVIEW OF RELATED WORKS

The detailed review of software vulnerability detection model on deep learning model is discussed and it is presented in Table 8. [33] aims for deep learning-based vulnerability detector for C programs source code. It offers an improvement of 9.8%, 7.9%, 8.2% on F-1 measure, FP-rate and FN-rate respectively when compared with [59]. The system uses intermediate representation of code and uses

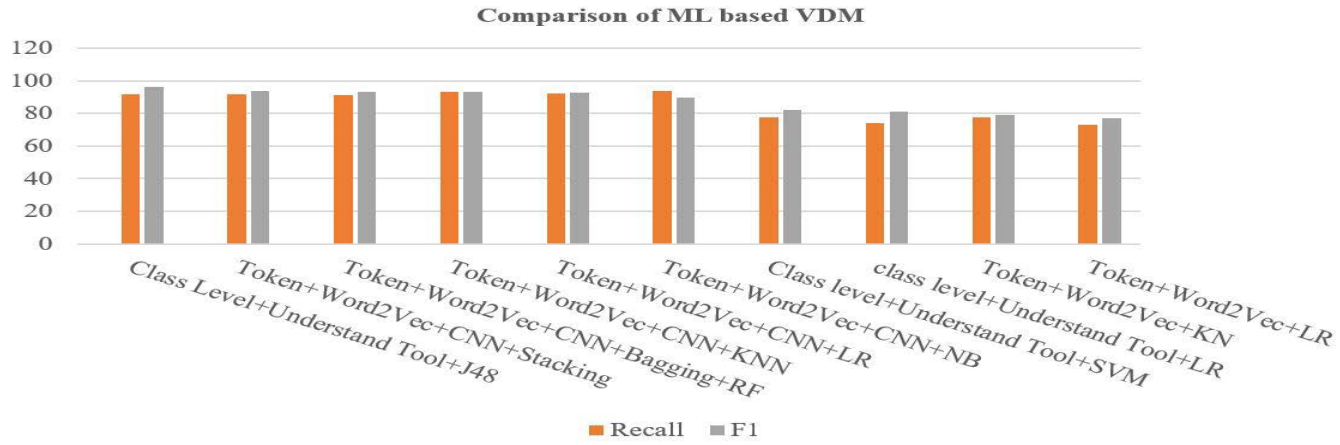

FIGURE 8. Comparison of ML VDM based on F1 and Recall score for the key aspects of VDM.

TABLE 1. Comparison of literature survey work with the proposed work.

Paper	ML	DL	GNN	Feature repre-senta-tion	Feature Vector-ization	Granularity	Dataset avail-ability	Label desc.	Vul. Types	Metric	Type of classi-fier
[5]	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓
[10]	×	✓	✓	✓	✓	✓	✓	✓	✓	×	×
[11]	×	✓	×	✓	×	✓	✓	✓	✓	×	×
[12]	✓	✓	×	✓	✓	×	✓	×	✓	✓	×
[13]	✓	×	×	✓	✓	×	×	✓	✓	×	✓
Proposed work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

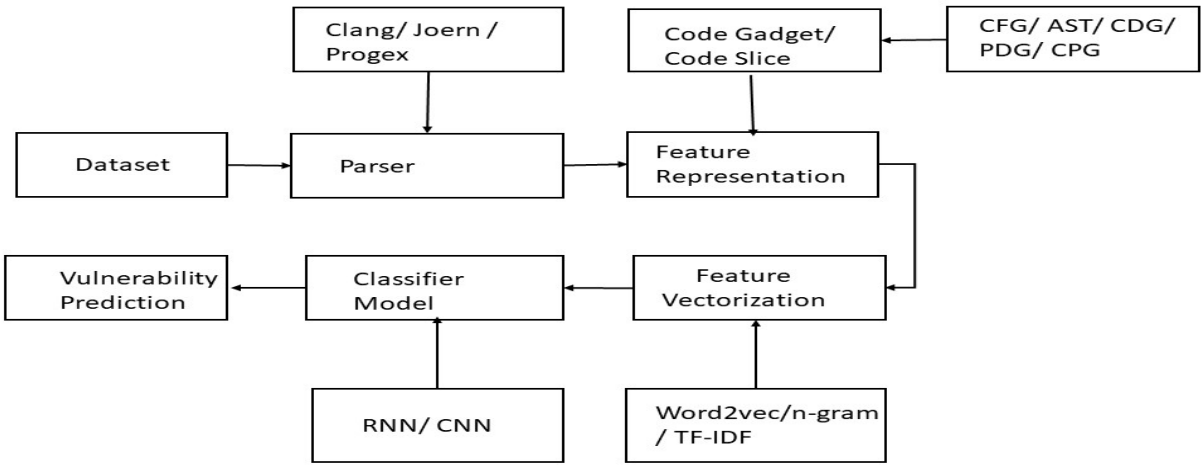

FIGURE 9. Represents specification of the key aspect of VDM based on DL model.

vulnerability syntax characteristics implemented in [59]. It further adds intermediate code representation component to the output of [59]. It is followed by labelling the intermediated code from as vulnerable or not, with line location precision. The work uses Abstract Syntax Tree (AST) [17] representation for retrieving syntax characteristics of vulnerabilities. This intermediate code is fed into compiler to generate intermediate representation (IR) files. It accommodates extra semantic information, uses the notion of granularity refinement to pin down locations of vulnerabilities. A word embedding method is used to the encode IR to vectors. The work has used Lower Level Virtual Machine(LLVM) [61] intermediate code and dg [62] to represent and define dependence relation between the IR files. The work uses Bidirectional LSTM(BLSTM) and Bidirectional GRU(BGRU) for accommodating preceding and

subsequent statements required for vulnerability detection. Further BRNN is extended with extra three layers of ie multiply, k pooling, and average pooling layer represented by ''vdl''. Thus BRNN-vdl achieves token level finer granularity with an F-1 measure of 97.22. To deal with imbalanced dataset it uses under-sampling method Near Miss-2 [63] and oversampling method SMOTE [64]. It uses NVD [1] and SARD [53] dataset. The disadvantage of the work is that it requires to compile source code into intermediate code, and cannot be used when a source code is not available or cannot be compiled. It only support C. It cannot accurately detect vulnerabilities that depend on dynamic information during program running. [59] is a vulnerability detection model for C/C++ source code programs. It detects 15 vulnerabilities from 4 datasets. The DL model is based on Bidirectional Gated Recurrent Unit (BGRU), which is more efficient than unidirectional RNN, CNN and even DBN. The system makes use of syntactic and semantic information through Abstract Syntax Tree (AST) [17] and Control flow Graph (CFG) [18]. CFG is defined using data dependency graph [18] and control dependency graph [18]. Program slices are retrieved from syntax and semantic retrieval of source. These slices are vectorized using user defined algorithm. It generates a binary classifier labelling vulnerable code as 1 and 0 otherwise (non-vulnerable). Also, the system is a binary classifier, thus it is not labelling what kind of vulnerability the code slice has, which in turn needs the intervention of human expertise to identify the specific vulnerability type which is a major limitation in this work. The reason for binary classification is attributed to code slice granularity achieved by the work, which results in imbalanced dataset. Its F1 is only 84.4% and it can be attributed that the work does not consider important code characteristics such as complexity which correlate with vulnerabilities. The neural networks take a fixed size vector as input, but on the other hand, the token number in each source entity (slice or function) may be different which leads to loss of information. Though it has significant improvement upon [38], i.e. a F1- measure is 86% and false positive rate of 10.1% and false negative rate (FN-rate) of 12.2%. This is attributed to inaccurate analysis of cross file dependence analysis of user defined or system define header files.

[65] It uses BLSTM neural network to detect software vulnerability. The input to vulnerability detector is code gadget, which is set of lines which may not sequentially occur but may be semantically related. To transform these tokens into vectors word2vec tool [66] is used. It produces only binary classifier. It has finer granularity but lacks mapping to specific vulnerability type. It uses only data dependent features to capture vulnerability characteristic. In the work [58] author has tried to pinpoint the vulnerability location along with multiclass labelling of the vulnerabilities found. It captures both data dependent features as well as control dependent features to effectively contextual information referred to as global semantics. To further localize the vulnerability, code attention mechanism is also used to capture local semantics. These global and local semantics are fused together and fed into BiLSTM networks to produce a multi-class vulnerability classifier. Code attention is an additional feature adopted from the work [65]. Code slice captures data dependency relation and control dependency relation. Code attention captures information specific to statement indicating arguments/parameters in a function call or API call. The system modelled using Bidirectional Long Short-Term Memory (BiLSTM) model. [65] detects vulnerability at multiple levels of line. But the system discovers vulnerabilities of only API/library calls. It uses only semantic information generated from data dependency graph. The work uses only one DL model based on RNN i.e. Bidirectional Long Short-Term Memory (BLSTM). Also, it targets only 2 types of vulnerabilities. [35] uses hybrid neural network i.e. combining CNN and RNN for efficient learning of local and global features for feature extraction and learning process. The source code is first converted to LLVM intermediate representation. This LLVM IR is in the form of Static Single Assignment(SSA). SSA provides explicit use-define chain and control dependency in the context. It uses Word2Vec embedding for vectorization of tokens. This is fed into hybrid neural network. It uses only SARD dataset for experimentation [53]. The work aims for inter-procedural statement level granularity. Thus, the vulnerability detector can detect in-project vulnerability. The work requires labelled dataset hence NVD [1] cannot be used since it provides only difference between the sample and patches. Thus, a rich labelled dataset is required for the working of this model. In draper [67] dataset labelling is done using static analysis which has problem of high false positive rate. Hence is not used for the performance evaluation of the work. The work is applicable to only C program source code which can be compiled. And not to other language, though the methodology may be adopted to other languages. [57]compares deep learning and machine learning algorithm and check the accuracy of vulnerability detected by different algorithm of each model. The work also compares different text vectorization technique which is used to generate vectors from code gadget. Code gadget is in turn generated using data flow analysis of the source code. [68] The system uses deep learning to extract the features and use deep learning algorithms to for prediction of vulnerabilities. The work uses C/ C++ source codes which is tokenized at slice level. It uses word embedding based on Fasttext using Fasttext tool [69]. But the system has slightly higher false positive rate compared to the obtained value of FNR and it not able to explain the reason for the same. Also, the work is only aiming for a binary classifier with code slice granularity. [70] is DL based classifier for VDM based on java source code. The method in itself has devised two types of models with different method for comparison. One model used AST to represent features and word2vec for feature vectorization. Whereas another model uses AST but BERT for feature vectorization. Thus, evaluated the performance of both the model. The system makes use SARD, NVD and Github which consist pf combination synthetic and semisynthetic code snippets. The work has used VulFind tool to label

the vulnerable code. Code from Github is mostly labelled via the tool. The work covers 118 CWE vulnerability types. [71] uses JavaLang Tokenizer [72] to tokenize the source code collected from Juliet dataset. This tokenized vector is fed into LSTM model to be classified. The work targets 29 type of vulnerability.

2) DISCUSSIONS

The most commonly used feature vectorization technique in DL VDM is Word2Vec. Code gadget and Graphical representation can both be used to represent features. The most commonly used algorithm in DL VDM is BLSTM. But its performance is not in par with [35] which uses a combination of CNN and RNN titled as Hybrid neural network. Moreover it is noted that instead of using graphical representation, code gadget representation performs better compared to Graph based feature representation. DL model is the only model in this literature review which have explored various vectorization techniques (count vectorizer, BERT, FastText). FastText Vectorization seems to give almost the same performance ad Word2Vec. Thus FastText can be further explored with other type of DL models to study its performance for VDM. This is derived from the Table 3 and is represented in graph using Figure 10.

C. KEY ASPECTS OF VDM ON GNN MODEL

Source code can be represented in many forms of graph like abstract syntax tree, control flow graph, data dependency graph etc. Thus work [31] is carried out on possibility of neural network to understand relationship between nodes and edges and can be extended for vulnerability detection system. Graph neural network (GNN) [25] has seen its implications in logic reasoning [73], code categorization [31], [74], variable prediction of code [75]. Thus, it was implicit to extend GNN in the field of cyber security domain of software vulnerability prediction. GNN is a class of DL models. It is an advanced variant of ANN model. GNN has work based on only C or only java projects. Whereas DL models are more proficient in C/C++ based java programs. The parser used in the existing work are open source tool namely, Joern' s [16] open source tool and clang tool [60] for C/ C++ programs. For generating graph for java source code open source tool, PROGEX [28]. It was designed and implemented based on ANTLRv4 [76] parser generator. PROGEX extracts graph such as AST, CFG and PDG with high speed in output formats like DOT, JSON and GML. Thus GNN based VDM represents features in using Graph based feature representation. For vectorization Word2vec, Doc2Vec or TF-IDF is used. But new vectorization technique such as Fasttext and GloVe can be explored to get an increase in accuracy rate in GNN based models. Much work is required in vulnerability detector model based on GNN to produce finer granularity at code slice or word level. The work reviewed here all are binary classifier which is again a tedious task for developer to find which vulnerability type the source code exhibit. The other advantage is that GNN and ML based VDM have mix of work based on both C and java projects, whereas DL needs to exploit its technique on java projects. The GNN explicitly focus on C based programming language and not on C++. Thus, future research work can focus on dataset with C++ source code and aim for word level granularity. Figure 11 represents a generalized working model of GNN based vulnerability detector with the techniques incorporated for key aspects in VDM model.

1) REVIEW OF RELATED WORKS

The detailed review of software vulnerability detection model on graph neural network model is discussed and it is presented in Table 9. [31] uses C programs source code from three datasets [67], [77], [78]. The features are presented using code property graph (CPG) [79]. CPG is a union of AST, CFG and PDG graph. Then word2vec is used to vectorize the graph nodes. Thus, the code graph vector is fed to the GNN model which learns vulnerabilities and establishes relationship between nodes and edges. The work uses Gated Graph Neural Network (GGNN) [73] model has shown good results in Juliet [77] and S-bAbL [78]. On draper dataset [67] F1 is 0.5 and average Precision is 0.4. The model compares with conventional ML model namely Random Forest (RF) and DL models like RNN and CNN. GGNN has performed better than both ML and DL models with F1 0.99 and F1 0.87 in S-bAbL and Juliet dataset respectively. The work proposed has not mentioned about granularity level which is open for future research work.

[32] The work aims to evaluate the effectiveness of GNN for program vulnerability analysis. The work has used both Graph Convolutional Network (GCN) [80] and Graph Attention Network (GAT) [81]. To evaluate GNN models, work is compared with static analysis tool like Spot Bugs [82] and conventional ML methods like Random Forest (RF) ensemble learning model. GCN has better accuracy compared to GAT since GAT has better learning capacity which can be used for learning cross-project analysis. Two vectorization technique is compared namely, TF-IDF and Doc2Vec [83]. Doc2Vec is an extension of word2vec which instead of word maps sentences, paragraph or full documents to target vectors. It is learned that TF-IDF is highly dependent on dimension. Smaller dimension size vector has better accuracy for TF-IDF which cannot be said for doc2vec model. The work has performed code normalization and compared the results and it is found that code normalization has negative effects on both GNN models. It is aligned with the work of [84]. It is found that for in- project analysis GCN has excellent performance than baseline model. GAT performs well for cross-project analysis since GCN suffers from overfitting problem. Also, TF/IDF is more reliant compared to Doc2Vec for in project analysis. System trains model for individual vulnerability and perform metric evaluation, which is not efficient because of time consumption requirement for training model on each type of vulnerability. Also, the author has not mentioned about the granularity level aimed for the proposed binary vulnerability detector. The advantage of the system is existing

TABLE 2. Comparison of performance of ML based VDM based on F1 and recall score.

SI.No.	Paper	Recall	F1 Metric	Key Aspect considered
1	[14]	91.66	96.3	Class Level+Understand Tool+J48
2	[34]	91.9	93.6	Token+Word2Vec+CNN+Stacking
3	[34]	91.2	93.1	Token+Word2Vec+CNN+Bagging+RF
4	[34]	93.4	93	Token+Word2Vec+CNN+KNN
5	[34]	92.1	92.9	Token+Word2Vec+CNN+LR
6	[34]	93.9	89.5	Token+Word2Vec+CNN+NB
7	[41]	77.4	82.3	Class level+Understand Tool+SVM
8	[41]	74.2	81	Class level+Understand Tool+LR
9	[34]	77.7	79.1	Token+Word2Vec+KN
10	[34]	72.9	77.1	Token+Word2Vec+LR

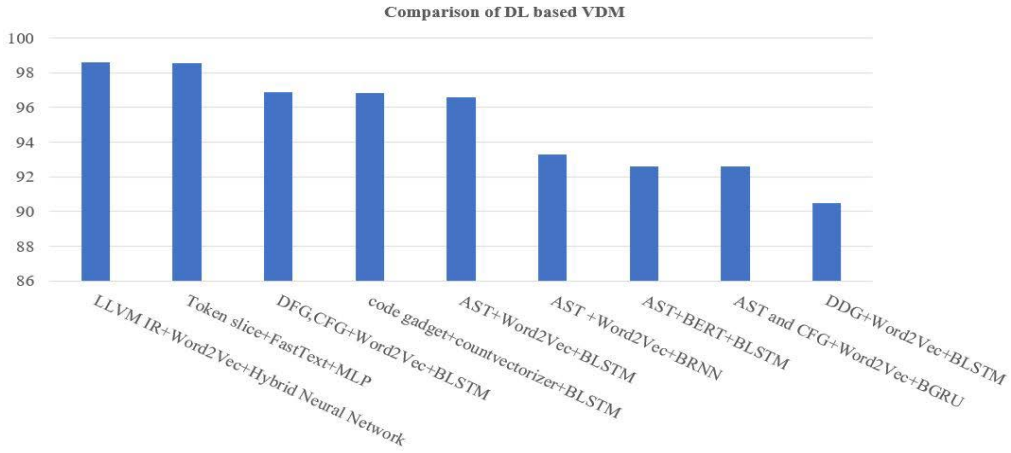

FIGURE 10. Comparison of DL VDM based on F1 score for the key aspects of VDM.

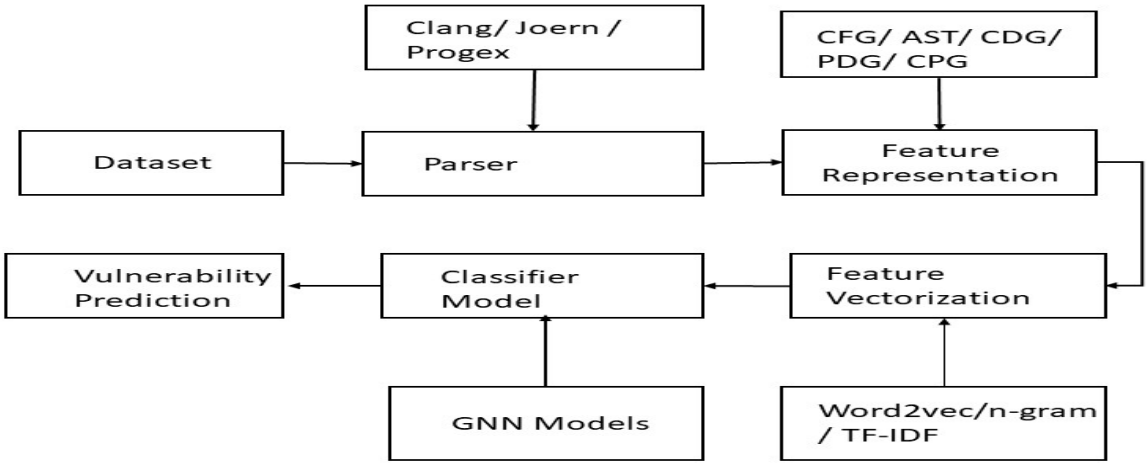

FIGURE 11. Represents techniques used in the key aspect of VDM based on GNN model.

work aim for C/C++ program, but the author has tried on java programming language dataset for which they have released an open source tool PROGEX [85] to generate AST, CFG, DDG for analysis.

[86] aims to address vulnerability at function level during version changes commit. The work feeds AST with node embedding and model GNN on it. The work uses GCN and GraphSAGE [80], [87] based GNN model. To evaluate the efficiency the work compares with CNN, SVM model, RF model. It uses Wireshark [88] dataset which is written in c programs and contains over 80 k commits. The system is validated only in one dataset and is limited to c programs.

TABLE 3. Comparison of performance of DL based VDM based on F1 score.

SI.No.	Paper	F1 Metric	Key Aspect considered
1	[35]	98.6	LLVM IR+Word2Vec+Hybrid Neural Network
2	[68]	98.55	Token slice+FastText+MLP
3	[58]	96.87	DFG,CFG+Word2Vec+BLSTM
4	[57]	96.82	code gadget+countvectorizer+BLSTM
5	[70]	96.58	AST+Word2Vec+BLSTM
6	[33]	93.3	AST+LLVM IR +Word2Vec+BRNN
7	[70]	92.61	AST+BERT+BLSTM
8	[59]	92.6	LLVM IR (AST & CFG & PDG)+Word2Vec+BGRU
9	[65]	90.5	DDG+Word2Vec+BLSTM

TABLE 4. Comparison of performance of GNN based VDM based on F1 score.

SI.No.	Paper	F1 Metric	Key Aspect considered
1	[31]	99	CPG+Word2Vec+GGNN
2	[95]	95.6	PDG,VFG,Program Slice+Word2Vec+kGNN
3	[29]	84.97	AST,CFG,DFG,NCS+Word2Vec+GGRN & MLP
4	[32]	84	CFG+Doc2Vec+GCN
5	[32]	77	CFG+Doc2Vec+GAT
6	[90]	76.8	AST, CFG,DFG+Word2Vec+BGNN
7	[86]	74.3	AST,CFG,DDG,CDG+Word2Vec+CNN
8	[86]	71	AST,CFG,DDG,CDG+Word2Vec+GraphSage
9	[9]	54.94	AST, CFG,DFG+Eord2Vec+3GNN & RF
10	[98]	36	PDG,CDG+CodeBERT+GAT
11	[98]	23.8	PDG,CDG+CodeBERT+GCN
12	[91]	NA	CDG,PDG+G-CodeBERT+GGNN
13	[91]	NA	CDG,PDG+G-CodeBERT+GCN
14	[94]	NA	CPG,NCS+Word2Vec+GAT

Thus, it can be implemented or tested on different dataset [1,24] for validity of the predictor. Moreover, system aims for binary classifier and granularity of token level. Thus, there is scope for increasing the granularity level and to come up with a multiclass classifier.

[29] model uses gated graph recurrent network (GGRN) model for learning the features which uses combination of AST, CFG, DFG and Natural code sequence (NCS). The model aims for function level granularity operating in C projects. First code is vectorized using Word2vec model and fed to GGRNN model and predictions are made using multi-layer perceptron (MLP). The work is done on manually labelled function dataset collected from 4 large C open source projects that are popular and diversified, namely Linux Kernel, QEMU, Wireshark and FFmpeg and is published for open access in [89]. Thus, this manual process consumed almost 600 Man hours to perform two round data labelling and cross-verification. With all this effort the work is only able to provide a coarse granularity at function level with a binary classifier. [90] suggest the use of Bidirectional Graph Neural-Network(BGNNN) by using sematic information constructed through AST, CFG, data flow graph (DFG) to represent features. CNN is used to further extract features.manually detects 2149 vulnerabilities and 3867 vulnerable function. It collected C/C++ open source projects (i.e. Linux Kernel, FFmpeg, WireShark, and Libav) from two sources i.e NVD [1] and Github. The proposed work has achieved higher precision and accuracy. [9] utilizes word2vec embedding for encoding features extracted from AST, CFG, DFG. The work is based on 3GNN which uses Crystal Graph Convolutional Networks (CGCN) and Self Attention Graph(SAG) Pooling. The work makes use of Draper, QEMU+ FFmpeg dataset. Model has a comparison between BiLSTM, CNN, GGNN, 3GNN. The performance of the model is enhanced by squeezing Random Forest to the final extracted features. [91] uses dataset (i.e. QEMU and FFmpeg) from dataset [89]. Features are represented as raw tokenized source code with embedding generated from codeBERT [92] and Graph- codeBERT [93]. The model is trained using GCN and GGNN model. [94] uses Big-Vul dataset and D2A dataset. The source code is represented using SIR graph which combine code property graph(CPG) and natural code sequence(NCS). The model is trained using GAT with attention mechanism with an accuracy of 95%. [95] uses CFG and Value flow Graph(VFG) for representing source code. From the code graph program slice is interest is extracted and is vectorized using Doc2Vec and fed into kGNN. The parameter k is set to 3. The dataset used in SARD and two real world dataset namely redis [96] and lua [97]. [98] extracts control flow dependencies and data flow dependencies from C/C++ projects using joern tool. It achieves statement level granularity. The work has explored Doc2Vec, GloVe, CodeBERT vectorization techniques. GNN models used are GAT, GCN and 91 vulnerability types are learned to produce a binary classifier. It uses labelled datasets such as Fan [99], Reveal [100] FFMPeg+QEMU [89].

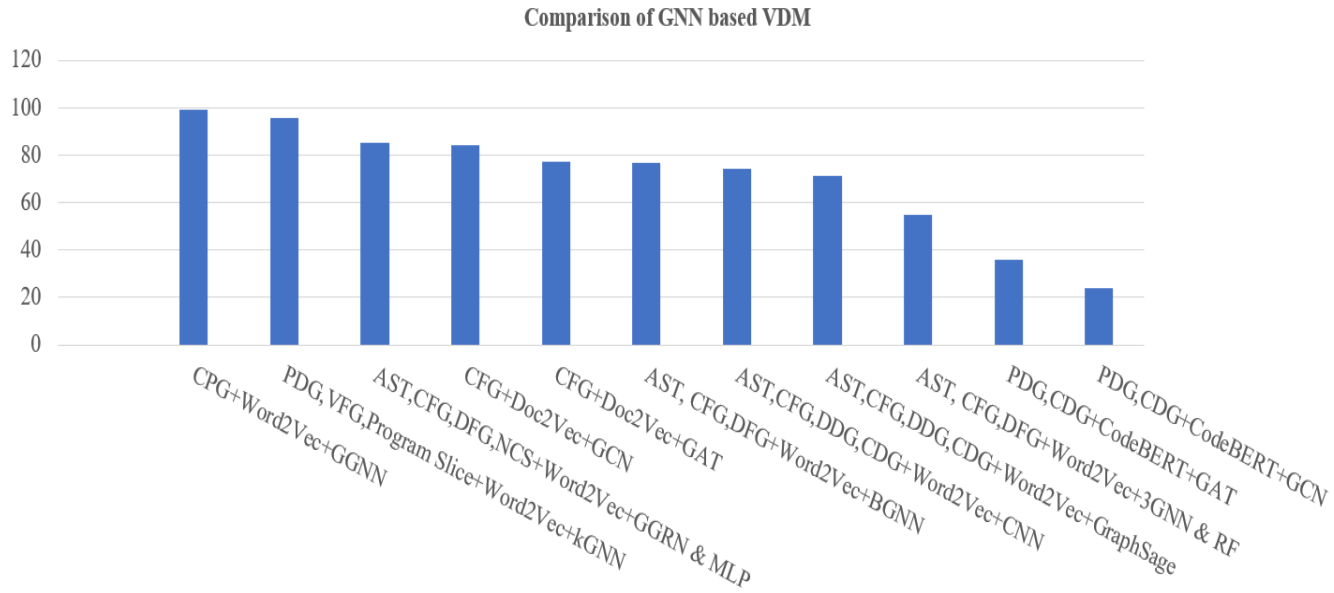

FIGURE 12. Comparison of GNN VDM based on F1 score for the key aspects of VDM.

TABLE 5. Vulnerability coverage provided by dataset for specific programming type.

Program. Type	Paper	Dataset	Max No. of Vul.	Paper	Dataset	Min No. of Vul
C/C++	[90]	NVD [1]+Github	2149	[95]	SARD [53] +redis [96]+lua [97]	10
C	[89]	Devign(Wireshark+ Linux+ FFMPeg+ QEMU) [89]	40	[31]	Juliet [77]+ Draper [67]+ s-bAbL [78]	5
Java	[70]	NVD [1]+SARD [53] +Juliet [77]+Github	118	[71]	Juliet [77]	29

2) DISCUSSION

In GNN based VDM, it is inevitable to carry out the vulnerability detection without extracting CPG of the source code. It can be interpreted from the comparison graph generated from considering F1 score of the reviewed articles. Comparison graph is shown in Figure 12. This graph is generate from Table 4. Other metrics are not considered since many work has not mentioned the Recall, accuracy or FPR or FNR. Hence the most used metric, using F1 score the graph is generated and these observations are derived. CPG is a combination of AST, CFG and PDF graph. Input to GNN model works better when feature representation is CPG. Most of work uses Word2Vec feature vectorization which has improved performance compared to vectorization technique based on BERT algorithm. In GNN VDM model, GGNN is performing better than most commonly used GNN model i.e. GAT and GCN. It can be attributed to the fact that the existing work has explored GAT and GCN with vectorization technique like CodeBERT and Doc2Vec. The lower performing model uses Code BERT algorithm for feature vectorization which uses PDG and CDG only. It can be concluded that GNN based VDM require CPG for feature representation. VDM can make use of Word2Vec or Doc2Vec as feature vectorization technique, CodeBERT doesnot seem to perform better for GNN model. For GNN based VDM model, GGNN works the best and GCN performs the worst. Most commonly used GNN models are GAT, GCN further work needs to be explored with word2vec and doc2vec feature vectorization technique and their performance need to be evaluated.

D. REVIEW OF DATASET CONSIDERED FOR TARGETED VULNERABILITIES

The proposed work has selected literature which has focused on Java and C/C++ and C program based open source projects. For C/C++ or java programs, the dataset used is based on National Vulnerability Database (NVD) [1] and Software Assurance Reference Dataset (SARD) [53] dataset. Both have a labelled dataset of source code and its corresponding vulnerability type. The details of the dataset used for the considered work is listed in Table 10. The table presents the dataset link, type of programming language considered, whether the dataset contains synthetic or semi-synthetic or real world projects. The Labelled field indicates that the dataset is labeled using static analysis tool or is labelled with predefined pattern or manually labelled using human expertise.

1) DATASET FOR C/C++ PROGRAMMING BASED PROJECTS

NVD contains vulnerabilities of software in production, whereas SARD is a standard dataset used to test vulnerability

TABLE 6. Research gaps in the literature's considered for the study.

SI.No.	Paper	Research Gaps
1	[65]	1. The work is effective in capturing vulnerabilities related to library/ API function calls, thus the vulnerabilities not associated to library /API function calls need to be targeted and effectiveness need to be evaluated. 2. Code gadget can be generated only though commercial tool, hence difficult to replicate. 3. Work captures only data flow dependency to generate code gadget. To generate the same capturing control flow dependency, need to be evaluated.
2	[59]	1. The work uses vulnerability syntax characteristic which maps attributes of nodes generated from AST of source code to vulnerability characteristic using joern and checkmarx tool. But it covers only 93.6 % vulnerable programs in the considered dataset. Thus other method can be explored for more vulnerability coverage. 2. The work achives code slice granularity but finer granularity can be achieved. 3. SARD dataset is synthetic thus have high degree of similarity between different cases. Thus ,classifier is biased.
3	[29]	1.It presents results on its own dataset which is only half released. Thus, other work cannot use their published dataset. 2. The work has not evaluated the performance on already existing and published database.
4	[33]	1.work target only programs written in C. To extract features tools used only support C. Thus, it need to be explored in other language type. 2.Source code which cannot be compiled cannot be used for this work since it required code to be converted into intermediate representation.
5	[71]	Work targets only 29 type of vulnerabilities out of the available 112 vulnerability types in the dataset, stating reduction of accuracy is not experimentally presented in the paper.
6	[90]	Work should have targeted vulnerabilities related to library API function call, Array Usage, Pointer Usage and arithmetic expression which [59] targeted. Then performance should have been evaluated instead of targeting all the vulnerabilities. 2. Also the work manually labelled the dataset based on CVE 2020 report
7	[9]	The work has trained on all Joern edges. The work could have explored on training on subset of edges based on work [29] and evaluated the performance difference.
8	[98]	Recall metric computed is very low, the work could have explored other vectorization techniques and compared.
9	[94]	The work has only presented the accuracy and not presented recall or F1 score without any explanation.
10	[91]	1. The work has explored the possibility of use of CodeBERT, but performance evaluation using Word2vec or Doc2Vec could have evaluated. 2. The work has only presented accuracy metric, the absence of metric such as F1 and Recall is not explained.
11	[95]	1.The real word dataset is manually labelled thus is open to human errors. No tool was used to verify the labelling. 2.The work was not evaluated in open source published dataset but used synthetic SARD dataset. 3. The work has targeted only 10 vulnerability type.
12	[70], [71]	The work is limited to using AST & BERT for feature representation and vectorization. LLVM IR [105] can be explored to use for java projects and performance can be compared.
13	[35]	1. Though the work has highest F1 score, but due to normalization of custom defined functions and identifiers to unified names it only achieve r a binary classifier, whereas the work on similar idea [33] achieve a multi-class classifier.Thus the work can be explored to produce a multi-class classifier. 2. The work only uses SARD data set because of availability of lack of labelled dataset. 3. It cannot use NVD dataset because of the limitation in vulnerability granularity that dataset provides. 4. Draper VDISC dataset generated using static analysis tool have the problem of high false positive rate. Thus labelled dataset for C program is still an open challenge. 5 .the work can use count vectorization technique and evaluate the performance to compare with [57] .
14	[58]	The work focus on vulnerabilities related to library/ API function calls, thus the vulnerabilities not associated to library /API function calls like resource management errors, improper resource lifetime control need to be targeted and explored.
15	[68]	1. The targeted vulnerability type is limited to Function call (FC), Array Usage (AU), Pointer Usage (PU), Arithmetic Expression (AE). 2. Other vulnerability type like buffer error vulnerabilities (CWE 119), resource management error vulnerabilities (CWE 399), and improper resource lifetime control (CWE 664) can also be targeted to improve vulnerability coverage. The listed vulnerabilities are considered as part of work [57].
16	[57]	1. The work has stated that replacement of user defined variables and function names is not helpful for vulnerability detection which is contrary to work [35] which has 98.6 F1 score . 2. [35] uses LLVM IR +word2vec. Thus, both method can be compared on some common ground and analyze the statement whether replacing name of custom variables and functions have its impact in vulnerability detection or not.

detection tools on software errors. In NVD each vulnerability is coined with unique identifier called by the term common vulnerabilities and exposure identifier (CVE ID). Vulnerabilities logged in SARD is identified by the term common weakness enumeration identifier (CWE ID). The two most common vulnerabilities considered from dataset NVD and SARD for the research works, [34] and [65] are buffer overflow identified with CWE-119 and resource management error (CWE-399). [33], [59], [68] uses collected and labelled C/C++ program source code from [43], [45]

TABLE 7. Key techniques used in VDM based on ML model.

Paper	Program lan	No. of vulner-ability	Granularity	Feature repre-senta-tion	Feature Vectorization	ML Model	Accurate Model	Type of classi-fier	Metrics used
[41]	C/C++ and Java	16	Class level, function level	Class level and func-tion level metrics	Understand Tool	SVM& LR	SVM	Binary	ROC curve, R, P, FPR, F mea-sure
[14]	Java	3	Project, pack-age, method, class level	Class level and func-tion level metrics	Understand Tool	32 ML algo-rithm	J48, AdaBoostM1 and LWL	Binary	ROC, Accu-racy, Kappa statis-tic, R, P, TP, FP, F mea-sure
[34]	C/C++	6	Word level	code slice	Word2Vec & CNN & LSTM	LR & MNB & RF	NA	Binary	R, P, FPR, FNR, F1

TABLE 8. Key techniques used in VDM based on DL model.

Paper	Program lan	No. of vulner-ability	Granularity	Feature repre-sentation	Feature Vectorization	DL Model	Type of classi-fier	Metrics used
[33]	C	18	Token	AST & LLVM IR	Word2vec	BRNN-vdl & BRNN	Multi-class	P, FPR, FNR, A, F1, IoU
[59]	C/C++	15	Code slice	LLVM IR (AST & PDG & CFG)	Word2Vec	BLSTM & BGRU	Binary	FPR, FNR, A, P, F1, MCC.
[65]	C/C++	2	Code gadget/slice	Code gadget	Word2vec	BLSTM	Binary	FPR, FNR, TPR, P, F1
[58]	C/C++	40	Code gadget	Code gadget and Code at-tention	Word2Vec	BLSTM	Multi-class	FPR, FNR, A, P, F1
[35]	C	12	inter-procedural statement-level	LLVM IR	Word2Vec	Hybrid neural net-work	Binary	FPR, FNR, A, P, F1
[57]	C/C++	3	Code gadget	Code gadget	Count vectorizer	DL & ML	Binary	P,FPR, FNR
[68]	C/ C++	4	Code slice	Tokenizing Slice	Fasttext	MLP	Binary	F1,P,FPR, FNR
[70]	Java	118	Token	AST	BERT& Word2Vec	BiLSTM	Binary	A,P,FPR, FNR
[71]	Java	29	Method -level	Javalang Tok-enizer	Javalang Tokenizer	LSTM	Binary	A,P,FPR, FNR

which is publicly available in SySeVR dataset [54]. The collected instances contain labelled instances with vulnerability type such as Function call (FC), Array Usage (AU), Pointer Usage (PU), Arithmetic Expression (AE). [57] also makes use of dataset [43], [45] for C/C++ programs targeting buffer error vulnerabilities (CWE 119), resource management error vulnerabilities (CWE 399), and improper resource lifetime control (CWE 664). [35] uses only SARD dataset and uses 11 type of vulnerabilities' for training the model. These are CWE-20, CWE-78, CWE-119, CWE-121, CWE-122, CWE-124, CWE-126, CWE-127, CWE-134, CWE-189 and CWE-399. [58] also uses SARD and NVD dataset with a vulnerability coverage of 40 classes. C/C++ open source datasets [97] & [96], which contains real-world projects are not labelled. These datasets are used in [95] are manually labelled with the help of experienced software engineers. These datasets mostly have 10 common types of vulnerabilities namely CWE-119, CWE-20, CWE-125, CWE-190, CWE-22, CWE-399, CWE-787, CWE-254, CWE-400, CWE-78. [98] uses combination of dataset Fan

TABLE 9. Key techniques used in VDM based on GNN model.

Paper	Program lan	No. of vulner-ability	Granularity	Feature represen-tation	Feature Vectoriza-tion	GNN Model	Type of classi-fier	Metrics used
[31]	C	5	Function level	Code Property Graph	Word2vec	GGNN	Binary	F1, P, R
[32]	Java	7	Program Control	Flow Graph	Doc2Vec or TF-IDF	GAT, GCN	Binary	P, R, F1
[86]	C	NA	Function level	AST, CFG, DDG, CDG	Doc2Vec and TF-IDF	Graph SAGE	Binary	R, P, F1
[29]	C	40	Function	AST, CFG, DFG, NCS	Word2vec	GGRN and MLP	Binary	A, F1
[90]	C/C++	2149	Function	AST, CFG, DFG,	Word2vec	BGNN	Binary	R,A, F1
[9]	C/C++	40	Function	AST, CFG, DFG,	Word2vec	3GNN	Binary	A, F1,MCC
[98]	C/C++	91	Statement level	PDG, CDG	Doc2Vec, GloVe, Code-BERT	GAT,GCN	Binary	R,P, F1
[91]	C/C++	40	function level	PDG, CDG	CodeBERT, Graph-codeBERT	GGNN,GCN	Binary	A
[94]	C/C++	60	function level	CPG & NCS	word2vec	GAT	Binary	A,P,R,F1, FPR, FNR
[95]	C/C++	10	function level	CFG CPG & VFG	Doc2Vec	kGNN	Binary	A,P, R,F1, FPR, FNR

[99], Revealchakraborty2021deep and FFMPeg+QEMU. FFMPeg+QEMUis already a part of dataset released by [89]. It is used by work [90] along with draper dataset [67]. [94] uses D2A dataset [101] and Big-Vul dataset [99]. D2A consist of source code of real world projects and labelled using static analysis tool, Infer [102]. Big-Vul dataset consist of 91 vulnerabilities collected from open source projects and CVE database.

2) DATASET FOR JAVA PROGRAMMING BASED PROJECTS

For collection of java source code programs, work [14] used source codes downloaded from online Github repository. [14] targets 3 types of vulnerabilities denoted by LawofDemeter (V1), BeanMemberShouldSerializeVulnerability(V2) and LocalVariableCould-BeFinalVaraible (V3) published in dataset [50]. Collected source codes are examined using Programming Error Detector Tool (PMD) for labelling the collected source code. PMD tool comes as Eclipse Plugin which detect vulnerability in the source code which in turn is fed as input to the model expected to train. [41] collects Java source code from open source projects like Apache release [42], [45] and Stanford SecuriBench dataset [43]. The vulnerability reported by Apache Tomcat report have uniqe CVE-ID, version number, fixed version details. Some of the vulnerabilities considered are SQL injections, Information disclosure, Denial of Service, Cross-site Scripting, Arbitrary file deletion. The work considered a total of 91 vulnerable classes by combining the projects [42], [45] and [43]. [32] which is a GNN based VDM is based on java source code targeting web application vulnerabilities. The OWASP Benchmark Project [103] is a synthetic java test suite, and has two version v1.1 and v1.2 consisting of 11 classes of labelled vulnerabilities. [32] targets 7 most common type of vulnerabilities namely, CWE-22, CWE-78, CWE-79, CWE -89, CWE-90, CWE-501, CWE-643 which denotes Path Traversal, Command injection, Cross-site Scripting, SQL injection, LDAP injection, Trust Boundary Violation, XPATH Injection attack.

3) DATASET FOR C PROGRAMMING BASED PROJECTS

GNN based VDM [29] is mostly based on C projects are retrieved from open source datasets such as Wireshark [88] dataset and Devign [89] dataset. Devign consist of vulnerabilities collected from 4 open source C librabries namely Linux Kernel, QEMU, Wireshark and FFmpeg. Thus, [86] has only limited its dataset on Wireshark and hence can be extended on test other 3 libraries. [31] collected C program codes from Juliet [77], Draper dataset [67] and s-bAbL [78]. Juliet consist of synthetic dataset with labelled classes as good or bad for non-vulnerable and vulnerable type. Draper dataset consist of synthetic and well as codes from open source projects like Debian and Github projects. It has 5 different vulnerability

TABLE 10. Details of dataset for programming languages considered for the study.

Dataset	Progr. Type	Real projects	Labelled	Tool for Labelling
NVD [1]	C/C++	✓	×	[65]
NVD [1]	Java	semisynthetic	×	VulFind [70]
SARD [53]	Java	semisynthetic	×	VulFind [70]
SARD [53]	C/C++	✓	✓	NA
SySeVR [54]	C/C++	✓	✓	NA
Apache release [42], [45]	C/C++	✓	✓	NA
Github	Java	✓	×	PMD & VulFind [70]
SecuriBench [43]	Java	×	✓	PMD
Juliet [77]	C Programs	×	✓	NA
Juliet [77]	Java	×	✓	Custom Method [71]
Draper [67]	C Programs	✓	✓	NA
s-bAbl [78]	C programs	×	✓	NA
Wireshark [88]	C programs	✓	✓	NA
Devign [89]	C programs	✓	×	Manual Labelling
OWASP Benchmark Project [103]	Java	×	✓	NA
redis [96]	C/C++	✓	×	Manual
lua [97]	C/C++	✓	×	Manual
Fan [99]	C/C++	✓	✓	NA
Reveal [100]	C/C++	✓	✓	NA
FFMPeg+ QEMU [89]	C/C++	✓	✓	NA
D2A [101]	C/C++	✓	×	Infer [103]
Big-Vul [99]	C/C++	✓	✓	NA

classes, CWE-119, CWE-120, CWE-469, CWE-476 and others (combines other CWEs). s-bAbl dataset is a synthetic dataset solely for buffer overflow vulnerabilities.

4) DISCUSSIONS

Based on the type of programming languages the datasets that support maximum and minimum vulnerability coverage is listed in Table 5 along with vulnerability type and number of vulnerability targeted.

IV. SUMMARY OF RESEARCH GAPS

Research gaps while reviewing the literature work is listed in Table 6.

V. CONCLUSION

For vulnerability detector system the most important requisite are availability of labelled dataset, identify which technique to use for feature representation and feature vectorization, expected granularity detection for the vulnerability to be detected and finally which algorithm should be used to model the classifier.

For feature representation, instead of using vectorized token as input which do not capture the context of the statement it is better to use representations which not only capture semantic, but also structural information. Thus, representation using graphs like LLVM IR, AST, CPG, PDG is the need of the hour [16], [24]. To provide more information to the models under consideration it is also suggested to use word embedding technique instead of just using frequency-based representation. Embedding provide high level source code representation while preserving contextual dependency. However, frequency-based representation captures insufficient information on semantic meaning and contextual dependence of each term. Embedding technique like word2vec and countvectorizer, fasttext showed better performance compared to BERT and understand tool.

The system should aim for finer granularity level as it lessens the interference of human experts which saves time. Thus, developers can concentrate more on fixing the vulnerability instead of reviewing the code to look for which code has caused vulnerability. ML model can achieve a maximum granularity of word level by using deep representation learning. The other granularity varies from class level to function level to project and package level. DL provides the fines granularity i.e. at token level. GNN model shows the best granularity at function level. Statement level granulairty showed by GNN model has very low F1 metric.

ML based VDM have mix of work on java and C/C++ source codes. DL lacks work on Java based source code which can be taken up as future work. Thus, DL based VMD can be explored on open source java dataset [42], [45], [43], [103]. GNN has mix of work in C program and C/C++ program based dataset [1], [53], [54] to check its credibility of stated accuracy. GNN models need to be experimented more in java based dataset, it can be attributed to the fact that there is no real world java projects based dataset available which is also labelled. Thus creating a well labelled java dataset for real world project is also an open problem.

The advanced ML and ANN model like DL and GNN based vulnerability models have achieved higher vulnerability coverage, finer granularity and less false positive. But the reviewed work could achieve only 89 to 91 % Precision. Thus, there is enough work to be done to improve upon the precision and recall while focusing finer level granularity at code slice or token level.

The ML based VDM can be used where coarse granularity is sufficient and provides a binary classifier. GNN model can be used when the requirement is for finer granularity which provides a binary vulnerability classifier. The vulnerability detector models reviewed have multiclass vulnerability detector only in DL based model with finer granularity. GNN based model which is still in infancy stage can explore on providing

a multiclass vulnerability detector since aim of automated vulnerability detector system is to reduce human intervention and less dependency on human expertise.

REFERENCES

[1] *National Vulnerability Database*, Nat. Inst. Standards Technol., U.S. Dept. Commerce, Gaithersburg, MD, USA, Jul. 2009.
[2] *Common Vulnerabilities and Exposures*, U.S. Dept. Homeland Secur. (DHS), Cybersec. Infrastruct. Secur. Agency (CISA), Washington, DC, USA. Accessed: Mar. 18, 2022.
[3] *25+ Cyber Security Vulnerability Statistics and Facts of 2021*, Edgescan Smart Vulnerability Manage., New York, NY, USA, Jan. 2022. Accessed: Mar. 18, 2022.
[4] A. O'Driscoll, "2022 vulnerability stats report," Edgescan Smart Vulnerability Managementd, New York, NY, USA, Vulnerability Statist. Rep. 2021, Jan. 2022. Accessed: Mar. 18, 2022.
[5] 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.
[6] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Trans. Softw. Eng.*, early access, Jun. 8, 2021, doi: 10.1109/TSE.2021.3087402.
[7] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*.
[8] A. Menzli, "Graph neural network and some of GNN applications: Everything you need to know," Neptune Labs, Tech. Rep., Dec. 2021. Accessed: Mar. 18, 2022.
[9] Y. Zhuang, S. Suneja, V. Thost, G. Domeniconi, A. Morari, and J. Laredo, "Software vulnerability detection via deep learning over disaggregated code graph representation," 2021, *arXiv:2109.03341*.
[10] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: A survey," *Proc. IEEE*, vol. 108, no. 10, pp. 1825–1848, Oct. 2020.
[11] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang, "Software vulnerability analysis and discovery using deep learning techniques: A survey," *IEEE Access*, vol. 8, pp. 197158–197172, 2020.
[12] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of Android malware detection with deep neural models," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 1–36, Nov. 2021.
[13] J. Jiang, X. Yu, Y. Sun, and H. Zeng, "A survey of the software vulnerability discovery using machine learning techniques," in *Proc. Int. Conf. Artif. Intell. Secur.* New York, NY, USA: Springer, 2019, pp. 308–317.
[14] A. Gupta, B. Suri, V. Kumar, and P. Jain, "Extracting rules for vulnerabilities detection with static metrics using machine learning," *Int. J. Syst. Assurance Eng. Manage.*, vol. 12, no. 1, pp. 65–76, Feb. 2021.
[15] *Checkmarx*, Checkmarx Limited, Ramat Gan, Israel. Accessed: Mar. 18, 2022.
[16] T. Reps, "Program analysis via graph reachability," *Inf. Softw. Technol.*, vol. 40, nos. 11–12, pp. 701–726, Dec. 1998.
[17] R. E. Noonan, "An algorithm for generating abstract syntax trees," *Comput. Lang.*, vol. 10, nos. 3–4, pp. 225–236, Jan. 1985.
[18] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, 1987.
[19] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," in *Proc. 3rd Annu. Symp. Document Anal. Inf. Retr. (SDAIR)*, vol. 161175. Princeton, NJ, USA: Citeseer, 1994, pp. 1–14.
[20] Y. Zhang, R. Jin, and Z. Zhou, "Understanding bag-of-words model: A statistical framework," *Int. J. Mach. Learn. Cybern.*, vol. 1, nos. 1–4, pp. 43–52, Dec. 2010.
[21] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Documentation*, vol. 60, no. 5, pp. 493–502, Oct. 2004.
[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.
[23] Y. Goldberg and O. Levy, "word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method," 2014, *arXiv:1402.3722*.
[24] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.
[25] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.
[26] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[27] T. Zimmermann, N. Nagappan, and L. Williams, "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.
[28] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," in *Proc. Symp. Bootcamp Sci. Secur.*, Apr. 2015, pp. 1–9.
[29] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "*Devign*: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.
[30] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Trans. Softw. Eng.*, vol. 40, no. 10, pp. 993–1006, Oct. 2014.
[31] S. Suneja, Y. Zheng, Y. Zhuang, J. Laredo, and A. Morari, "Learning to map source code to software vulnerability using code-as-a-graph," 2020, *arXiv:2006.08614*.
[32] S. M. Ghaffarian and H. R. Shahriari, "Neural software vulnerability analysis using rich intermediate graph representations of programs," *Inf. Sci.*, vol. 553, pp. 189–207, Apr. 2021.
[33] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu, and H. Jin, "VulDeeLocator: A deep learning-based fine-grained vulnerability detector," *IEEE Trans. Depend. Sec. Comput.*, vol. 19, no. 4, pp. 2821–2837, Jul. 2022.
[34] L. Wang, X. Li, R. Wang, Y. Xin, M. Gao, and Y. Chen, "PreNNsem: A heterogeneous ensemble learning framework for vulnerability detection in software," *Appl. Sci.*, vol. 10, no. 22, p. 7954, Nov. 2020.
[35] X. Li, L. Wang, Y. Xin, Y. Yang, Q. Tang, and Y. Chen, "Automated software vulnerability detection based on hybrid neural network," *Appl. Sci.*, vol. 11, no. 7, p. 3201, Apr. 2021.
[36] C. J. Willmott, S. G. Ackleson, R. E. Davis, J. J. Feddema, K. M. Klink, D. R. Legates, J. O'Donnell, and C. M. Rowe, "Statistics for the evaluation and comparison of models," *J. Geophys. Res., Oceans*, vol. 90, no. C5, pp. 8995–9005, 1985.
[37] N. Chinchor and B. M. Sundheim, "MUC-5 evaluation metrics," in *Proc. 5th Message Understanding Conf. (MUC)*, Baltimore, MD, USA, Aug. 1993, pp. 1–10.
[38] A. Rosebrock, "Software assurance reference dataset (SARD) manual share," PyImageSearch, Tech. Rep., Nov. 2016. Accessed: Mar. 18, 2022.
[39] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica Biophysica Acta, Protein Struct.*, vol. 405, no. 2, pp. 442–451, 2016.
[40] *Understand™ Quick Feature List*, Sci. Toolworks, Hurricane, UT, USA. Accessed: Mar. 18, 2022.
[41] K. Z. Sultana, V. Anu, and T.-Y. 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, 2021, Art. no. e2303.
[42] *Apache Tomcat*, Apache Softw. Found., Forest Hill, MD, USA. Accessed: Mar. 18, 2022.
[43] B. Livshits, "Securibench micro," GitHub, Tech. Rep., Dec. 2014. Accessed: Mar. 18, 2022.
[44] *Early Security Vulnerability Detector*, Eclipse Found., Ottawa, ON, Canada. Accessed: Mar. 18, 2022.
[45] *Apache CXF™: An Open-Source Services Framework*, Apache Softw. Found., New Orleans, LA, USA. Accessed: Mar. 18, 2022.
[46] *Welcome to the Archives of the Apache Software Foundation!* Apache Softw. Found., Forest Hill, MD, USA. Accessed: Mar. 18, 2022.
[47] A. Verma, "Evaluation of classification algorithms with solutions to class imbalance problem on bank marketing dataset using WEKA," *Int. Res. J. Eng. Technol.*, vol. 5, no. 13, pp. 54–60, 2019.
[48] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
[49] W. A. Harrison and K. I. Magel, "A complexity measure based on nesting level," *ACM SIGPLAN Notices*, vol. 16, no. 3, pp. 63–74, 1981.

[50] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1143–1191, 2016.
[51] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, Jul. 2018.
[52] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. Int. Conf. Eng. Technol. (ICET)*, Aug. 2017, pp. 1–6.
[53] *Software Assurance Reference Dataset (SARD) Manual Share*, Nat. Inst. Standards Technol. (NIST), U.S. Dept. Commerce, Gaithersburg, MD, USA. Accessed: Mar. 18, 2022.
[54] J. Vaidya, *SySeVR: A Framework for Using Deep Learning to Detect Vulnerabilities*, Rutgers Univ., Newark, NJ, USA. Accessed: Mar. 18, 2022.
[55] C. Dyer, A. Kuncoro, M. Ballesteros, and N. A. Smith, "Recurrent neural network grammars," 2016, *arXiv:1602.07776*.
[56] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.
[57] W. Zheng, J. Gao, X. Wu, Y. Xun, G. Liu, and X. Chen, "An empirical study of high-impact factors for machine learning-based vulnerability detection," in *Proc. IEEE 2nd Int. Workshop Intell. Bug Fixing (IBF)*, Feb. 2020, pp. 26–34.
[58] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "μVulDeePecker: A deep learning-based system for multiclass vulnerability detection," *IEEE Trans. Depend. Sec. Comput.*, vol. 18, no. 5, pp. 2224–2236, Sep./Oct. 2021.
[59] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "SySeVR: A framework for using deep learning to detect software vulnerabilities," *IEEE Trans. Depend. Sec. Comput.*, vol. 19, no. 4, pp. 2244–2258, Jul. 2022.
[60] *Clang: A C Language Family Frontend for LLVM*, LLVM. Accessed: Mar. 18, 2022.
[61] H. Liang, L. Wang, D. Wu, and J. Xu, "MLSA: A static bugs analysis tool based on LLVM IR," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, May 2016, pp. 407–412.
[62] M. Chalupa, "DG: A program analysis library," GitHub, Tech. Rep. Accessed: Mar. 18, 2022.
[63] I. Mani and I. Zhang, "kNN approach to unbalanced data distributions: A case study involving information extraction," in *Proc. Workshop Learn. Imbalanced Datasets (ICML)*, vol. 126, 2003, pp. 1–7.
[64] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.
[65] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "VulDeePecker: A deep learning-based system for vulnerability detection," 2018, *arXiv:1801.01681*.
[66] R. Rehuřek, "Word2Vec embeddings," 2018, *arXiv:1801.01681*.
[67] L. Kim and R. Russell, "Draper VDISC dataset—Vulnerability detection in source code," Center Open Sci., Tech. Rep., Sep. 2018. Accessed: Mar. 18, 2022. [Online]. Available: https://arxiv.org/abs/1807.04320v2
[68] M. Alenezi, M. Zagane, and Y. Javed, "Efficient deep features learning for vulnerability detection using character n-gram embedding," *Jordanian J. Comput. Inf. Technol.*, vol. 7, no. 1, pp. 1–15, 2021.
[69] *Fast Text Library for Efficient Text Classification and Representation Learning*, Meta Res., Toronto, ON, Canada, Apr. 2020. Accessed: Mar. 18, 2022.
[70] H. Zhang, Y. Bi, H. Guo, W. Sun, and J. Li, "ISVSF: Intelligent vulnerability detection against Java via sentence-level pattern exploring," *IEEE Syst. J.*, vol. 16, no. 1, pp. 1032–1043, Mar. 2022.
[71] N. Saccente, J. Dehlinger, L. Deng, S. Chakraborty, and Y. Xiong, "Project achilles: A prototype tool for static method-level vulnerability detection of Java source code using a recurrent neural network," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. Workshop (ASEW)*, Nov. 2019, pp. 114–121.
[72] C. Thunes, "Javalang," MIT License, Tech. Rep., 2013.
[73] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2015, *arXiv:1511.05493*.
[74] P. Vytovtov and K. Chuvilin, "Unsupervised classifying of software source code using graph neural networks," in *Proc. 24th Conf. Open Innov. Assoc. (FRUCT)*, Apr. 2019, pp. 518–524.
[75] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," 2017, *arXiv:1711.00740*.
[76] T. Parr, "ANTLR (another tool for language recognition)," ANTLR/Terence Parr, Tech. Rep., 2014. Accessed: Mar. 18, 2022.
[77] *Test Suites*, Nat. Inst. Standards Technol., U.S. Dept. Commerce, Gaithersburg, MD, USA, Jan. 2006. Accessed: Mar. 18, 2022.
[78] C. D. Sestili, W. S. Snavely, and N. M. VanHoudnos, "Towards security defect prediction with AI," 2018, *arXiv:1808.09897*.
[79] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 590–604.
[80] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[81] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.
[82] *Spot Bugs Find Bugs in Java Program*, Free Softw. Found. (FSF). Accessed: Mar. 18, 2022.
[83] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1188–1196.
[84] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–37, 2018.
[85] S. M. Ghaffarian, "Progex (program graph extractor)," GitHub, Tech. Rep., Apr. 2020. Accessed: Mar. 18, 2022.
[86] S. E. Şahin, E. M. Özyedierler, and A. Tosun, "Predicting vulnerability inducing function versions using node embeddings and graph neural networks," *Inf. Softw. Technol.*, vol. 145, May 2022, Art. no. 106822.
[87] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[88] G. Combs, "Wireshark: Network protocol analyzer," Wireshark, Tech. Rep., 1998. Accessed: Mar. 18, 2022.
[89] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu. (Oct. 29, 2021). *Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics Via Graph Neural Networks*. [Online]. Available: https://github.com/epicosy/devign
[90] S. Cao, X. Sun, L. Bo, Y. Wei, and B. Li, "BGNN4VD: Constructing bidirectional graph neural-network for vulnerability detection," *Inf. Softw. Technol.*, vol. 136, Aug. 2021, Art. no. 106576.
[91] V.-A. Nguyen, D. Q. Nguyen, V. Nguyen, T. Le, Q. H. Tran, and D. Phung, "ReGVD: Revisiting graph neural networks for vulnerability detection," 2021, *arXiv:2110.07317*.
[92] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.
[93] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. Kun Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "GraphCodeBERT: Pre-training code representations with data flow," 2020, *arXiv:2009.08366*.
[94] Z. Song, J. Wang, S. Liu, Z. Fang, and K. Yang, "HGVul: A code vulnerability detection method based on heterogeneous source-level intermediate representation," *Secur. Commun. Netw.*, vol. 2022, pp. 1–13, Apr. 2022.
[95] X. Cheng, H. Wang, J. Hua, G. Xu, and Y. Sui, "DeepWukong: Statically detecting software vulnerabilities using deep graph neural network," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 3, pp. 1–33, May 2021.
[96] *Redis*, Redis Ltd. Accessed: Mar. 18, 2022.
[97] *Luaspot*, Departamento de Informática, Rio de Janeiro, Brazil. Accessed: Mar. 18, 2022.
[98] D. Hin, A. Kan, H. Chen, and M. Ali Babar, "LineVD: Statement-level vulnerability detection using graph neural networks," 2022, *arXiv:2203.05181*.
[99] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A C/C++ code vulnerability dataset with code changes and CVE summaries," in *Proc. 17th Int. Conf. Mining Softw. Repositories*. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 508–512.
[100] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Trans. Softw. Eng.*, early access, Jun. 8, 2021, doi: 10.1109/TSE.2021.3087402.
[101] Y. Zheng, S. Pujar, B. Lewis, L. Buratti, E. Epstein, B. Yang, J. Laredo, A. Morari, and Z. Su, "D2A: A dataset built for AI-based vulnerability detection methods using differential analysis," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2021, pp. 111–120.
[102] *A Tool to Detect Bugs in Java and C/C++/Objective-C Code Before it Ships*, Facebook, Built With Docusaurus. Accessed: Mar. 18, 2022.
[103] OWASP Foundation. (2022). *OWASP Benchmark Project*. [Online]. Available: https://owasp.org/www-project-benchmark/
[104] A. C. Andrew and D. Zagieboylo, "JLang: Developer guide," Tech. Rep.

POOJA S received the master's degree in cyber security from the College of Engineering Trivandrum, Trivandrum, Kerala, India. She is currently pursuing the Ph.D. degree in cyber security with the Manipal Academy of Higher Education, Manipal, Karnataka, India. She also works as an Assistant Professor with the Department of Information Communication Technology, Manipal Institute Technology (MIT), Manipal Academy of Higher Education. Her research interests include blockchain, machine learning, deep learning, and cryptography.

CHANDRAKALA C. B. received the degree in electronics and communication engineering from the Sri Jayachamarajendra College of Engineering (SJCE), Mysore University, Mysore, Karnataka, India, the master's degree in technology specializing in software engineering from SJCE, and VTU, Karnataka, and the Ph.D. degree from the MAHE, Manipal. She has experience of working both in industry and academia. She is currently working as an Associate Professor-Senior with the Department of Information Technology Communication, Manipal Institute of Technology, MAHE. Her research interests include distributed computing, speech processing and recognition, blockchain technology, and software engineering.

LAIJU K. RAJU received the master's degree in cyber security from the College of Engineering Trivandrum, Trivandrum, Kerala. He is a QA Architect at dltledgers India Private Limited, Trivandrum, Kerala, India. His research interests include blockchain, computer vision, and information security.