中国矿业大学计算机学院

2020 级本科生课程报告

课程名称	信息安全新技术
报告时间	2023.9.26
姓　　名	周子欣
学　　号	12204202
专业班级	信息安全 20-3 班
任课教师	李昕　张艳群
	张爱娟　鲍宇

《信息安全新技术》课程报告评分表

序号	毕业要求	课程教学目标	考查方式与考查点	占比	得分
1	5.1	目标 1：能够检索信息安全的最新进展资料。	能够通过文献索引、网络数据库检索等途径全面、准确地检索并引用最近 3 年的参考资料。参考文献不少于 10 篇，全英文参考文献。考查点包括参考文献的相关性、完整性、及时性、参考文献引用是否符合规范。	20%	
2	10.1	目标 2：能够阅读信息安全英文文献，能够撰写文章的英文摘要。	能够阅读英文论文，能够撰写专题报告的英文摘要并符合规范。英文摘要不少于 200 个单词。考查点包括英文摘要是否通顺、摘要内容是否能够全面准确地概括所研究的专题。	30%	
3	10.2	目标 3：能够了解信息安全领域的热点问题、前沿问题，并形成自己的观点。	掌握综述写作要点，归纳整理已有文献，分析鉴别已有方法或方案，对所研究的专题总结已经取得的研究成果，指出存在问题以及发展趋势等，给出自己的评论。考查点包括专题的选题是否新颖、总结分析是否得当、指出的优缺点是否准确、能否提出问题并给出自己的解决思路。	50%	
总分				100%	

评阅人：

2023 年 10 月 10 日

Abstract

The increasing security vulnerabilities have become an important concern in the field of software industry and network security, which indicates that th e current vulnerability detection methods need to be further improved. The vi gorous development of open source software community makes a large numbe r of software codes available, which enables machine learning and data minin g technology to make use of rich patterns in software codes. In particular, the recent breakthrough application of deep learning in speech recognition and m achine translation shows the great potential of neural models in natural langua ge understanding. This has inspired researchers in the software engineering an d network security communities to apply deep learning to learning. In this pa per, we review the recent research on using deep learning to detect vulnerabil ities, aiming at explaining how they use the latest neural technology to captur e possible vulnerability code patterns. Later, we pointed out some interesting f indings in all the papers and the challenges that need to be overcome to mak e progress in this field.

Keywords: Deep learning; vulnerability detection; software vulnerability; repre-sentation learning

Software Vulnerability Detection Using Deep Learning Techniques

1 Introduction

With more and more industries using software as their business carrier, c ode has become one of the most basic elements to support the normal operati on of society, and the security of software is also becoming a fundamental an d basic problem in today's society. The quality of software has directly affect ed people's daily lives. Due to the exposure of software vulnerabilities, compu ter systems are being attacked by such attacks as buffer overflow, denial of s ervice (DoS) and XSSCross site script (XSS). The latest CVE report shows t hat the number of vulnerabilities collected in the first two quarters of 2022 in creased by 31.4% compared with the same period in 2021[1]; According to a technical report of the National Institute of Standards and Technology, the eco nomic loss caused by software defects in the United States is as high as $60 billion every year[2]. The study of using software defects or vulnerabilities be fore or after they appear is an important topic in the field of program securit y.

With the vigorous development of open source software, researchers can get more and more information about code and defects. Established in 2008, GitHub, an open source project hosting website, has more than 50 million use rs, 100 million open source projects and 200 million submission requests by 2020, covering more than 428 million files. By September 2021, NVD, a def ect directory website, has collected 171,178 pieces of defect data, which provi de a sufficient data base for the study of defect detection based on learning.

In recent years, the large-scale data and the increasing computing power of hardware have enabled deep learning technology to make breakthrough pro gress in many tasks in the fields of image processing, speech recognition, nat ural language processing, and contributed to the third wave of artificial intelli gence. Therefore, researchers have gradually begun to apply deep learning in t he code field to try to replace traditional research methods, hoping to solve s ome problems that cannot be solved by existing methods through deep learnin g.

After several years' exploration, the ability of deep learning to mine defe ct code features has been verified to some extent, and more and more researc hers have been attracted to use deep learning from traditional defect detection methods. At the same time, the application of deep learning method to sourc e code defect detection still faces many challenges in data set construction an d model design. This paper focuses on the research of software vulnerability detection technology based on deep learning.

1.1 Software Vulnerability Detection Based on Traditional M achine Learning

Aiming at the defect detection of large-scale complex software systems, r esearchers try to break through the bottleneck of traditional defect detection m ethods by optimizing and reforming existing methods, and on the other hand, try to explore new intelligent software defect detection methods. Intuitively, m ost defect-related information can be obtained through code analysis, that is, t he ability to mine software defects is closely related to the ability to analyze code data. Learning-based method is very suitable for discovering and learnin g rules from massive data. In theoretical research, Hindle[3] et al. compared pr ogramming languages with natural languages by statistical methods. The result s show that they have very similar statistical characteristics, and even the pro gramming languages are more regular, thus putting forward the hypothesis of "natural theory" of codes. Inspired by this hypothesis, researchers gradually re alized the feasibility of analyzing and generalizing the rules contained in code s by statistical learning methods.

Early research on defect detection based on learning mainly used machin e learning methods. For example, VCCFinder[4] designed artificial features for data morphology and made statistics to get differentiated feature items. After One-Hot coding the sample feature values, support vector machine was used t o classify them. However, this kind of method has great shortcomings in effic iency and effect: on the one hand, most machine learning methods still need experts to construct features as input, and its detection ability is limited by th e quality of feature engineering; On the other hand, the current machine learn ing method has a high false positive rate in the actual detection effect, and it is difficult to meet the needs of practical application. The fundamental reaso n is that the machine learning model has limited ability to mine deep feature s.

1.2 Software Vulnerability Detection Based on Deep Learni ng

Thanks to the great success of deep neural network in image recognition, natural language processing and other tasks, people have found that deep lea rning method has more advantages in mining deep features. Some researchers have begun to try to introduce it into source code defect detection tasks. The current results show that deep learning also has incomparable advantages in defect detection tasks compared with traditional methods and machine learning methods. Moreover, due to its relatively short research history, There is still a lot of room for researchers to explore. The core components of using deep learning to detect defects are two parts: defect code data set and deep learnin g defect detection model. The defect code data set is the learning foundation and feature source of deep learning model, and it needs to have enough code data to represent defect codes and non-defect codes. The deep learning defec t detection model needs to set up an appropriate network structure according t o the characteristics of codes, so as to fully tap the features in the defect co de data set and obtain the ability to distinguish defect codes from non-defect codes.

2 Related work

2.1 Code vector representation method

At present, many methods, including variable name and type, AST, progra m dependency graph and data dependency, have been tried to represent the sy ntax and semantic information contained in the code, as shown in Figure 1. I n addition to the single representation method, many researches have also foc used on multi-dimensional mixed representation methods, which have brought certain performance improvement to the model. Among them, AST can extract the syntax and structural features in the code well. Therefore, it is widely us ed in the code representation stage of various models, but how to embed the source code into the vector space without losing the grammatical structure c haracteristics and semantic information content, so as to better complete the c ode representation work, still needs researchers to continue to explore.

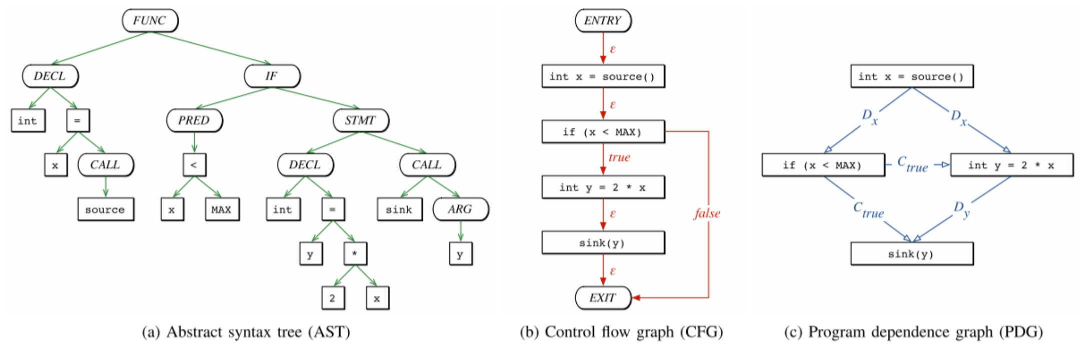


Fig. 1. Three methods of code vector representation

2.2 Deep learning method

Aiming at the problem of vulnerability detection, many researchers have d esigned different detection methods based on deep learning, mainly using GN N ,Transformer,CNN,RNN, LSTM and other structures. The following is a bri ef introduction to GNN ,Transformer and LSTM.

2.2.1 GNN

GNN (Graph neural network) is a kind of deep learning model with grap h structure data as input. It embeds the nodes and edges in the graph into a low-dimensional vector space for further processing and analysis. The core ide a of GNN is information transmission and aggregation based on local neighbo rhood information. GNN is a kind of neural network model developed in rece nt years, which is used to model and predict graph structure data and has ma de important applications in many fields. For example, in social network anal ysis, GNN can be used for tasks such as community discovery, node classific ation and mining influence subgraphs. In chemical molecular analysis, GNN c an be used for molecular representation, ligand screening and drug design. Fr om the perspective of computer vision and natural language processing, GNN can also be applied to image classification, object detection, emotion analysis and other tasks.

The whole GNN process is shown in Figure 2. Simply put, for each node, the embedded expression of the node is constantly updated by continuously aggregating the information of related nodes to the node. When this process is extended to the whole graph, the embedded expressions of all nodes on the whole graph can be updated, so that the characteristics of all nodes can be summarized and classified by using this characteristic.

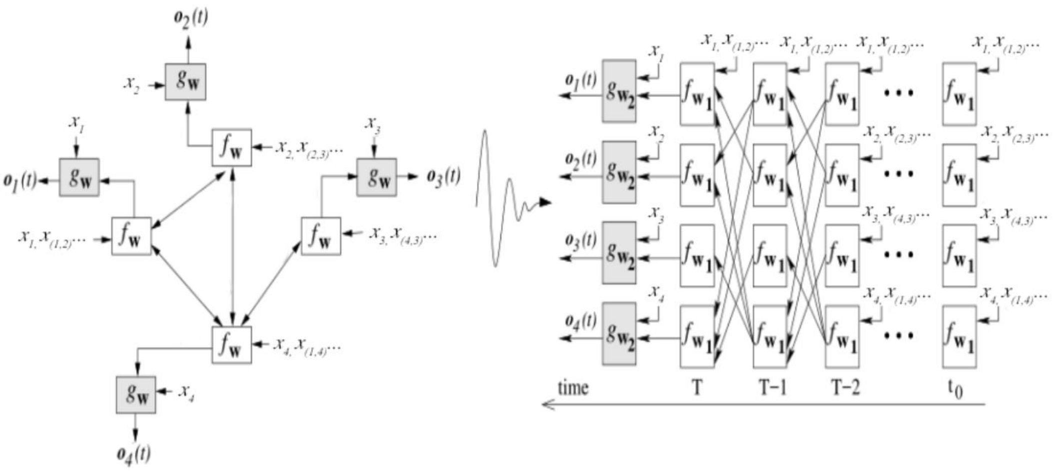


Fig. 2. GNN operation principle

2.2.2 Transformer

Transformer is a neural network model based on self-attention mechanism, proposed by Vaswani et al. in 2017. It has achieved significant breakthroughs in natural language processing tasks and has become a standard model in many NLP applications, such as machine translation, text generation, and language understanding. Unlike traditional recurrent neural networks (RNNs) or convolutional neural networks (CNNs), Transformer adopts a completely attention-based architecture, discarding the sequential nature of traditional models and enabling parallelization and efficiency.

The core idea of Transformer is to model the dependencies between different positions in the input sequence using self-attention mechanism. Self-attention allows the model to capture the interactions between each position and all other positions in the input sequence, leading to a better understanding of the contextual information. As shown in Figure 3, in Transformer, the input sequence is first embedded into a high-dimensional vector space and then passed through multiple layers of encoders and decoders. The encoder converts the input sequence into a series of semantically rich representations, while the decoder utilizes the encoder's output and self-attention mechanism to generate the target sequence step by step. The self-attention mechanism consists of querie s, keys, and values. By calculating the similarity between queries and keys, th e model can determine the attention weights for each position with respect to other positions and apply these weights to the values. This mechanism allow s the model to automatically allocate importance to different positions based o n the semantic features of the input sequence. In addition to self-attention me chanism, Transformer also introduces techniques such as residual connections and layer normalization to improve model stability and training effectiveness. Furthermore, Transformer uses positional encoding to introduce positional infor mation for different positions in the input sequence, enabling the model to ha ndle the order of the sequence.

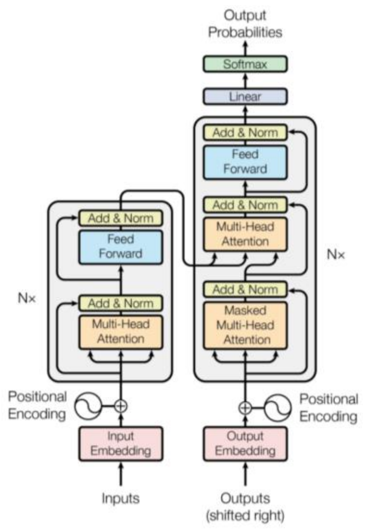


Fig. 3. Transformer structure

2.2.3 LSTM

LSTM (Long Short-Term Memory) is a special type of Recurrent Neural Network (RNN) architecture proposed by Hochreiter and Schmidhuber in 1997. Compared to traditional RNN, LSTM is capable of handling long sequential data more effectively and capturing and maintaining long-term dependencies.

LSTM structure, as shown in Figure 4, incorporates gate units to control information flow and retention. It consists of three types of gate units: input gate, forget gate, and output gate. The input gate determines which informatio n should be updated and incorporated into the cell state. It uses a sigmoid fu nction to produce a value between 0 and 1, indicating the importance of each input. Additionally, it employs a hyperbolic tangent function to generate a va lue between -1 and 1 as a candidate update for the input. The forget gate de cides which information from the previous time step in the cell state should be discarded. It uses a sigmoid function to generate a value between 0 and 1, indicating the extent to which certain parts of the cell state should be forgot ten. The output gate determines the hidden state to be output based on the c urrent cell state. It uses a sigmoid function to determine which parts of the c ell state should be output, and employs a hyperbolic tangent function to gener ate a value between -1 and 1 for producing the hidden state. Through the col laboration of these gate units, LSTM can adaptively update the cell state base d on the input sequence and previous memory, and generate meaningful hidde n states for the current task.

LSTM has achieved excellent performance in various tasks, especially whe n dealing with sequential data in natural language processing tasks such as la nguage modeling, machine translation, and text generation. It effectively addre sses the problems of vanishing and exploding gradients, while capturing and maintaining long-term dependencies, enabling the model to better understand a nd generate sequential data.

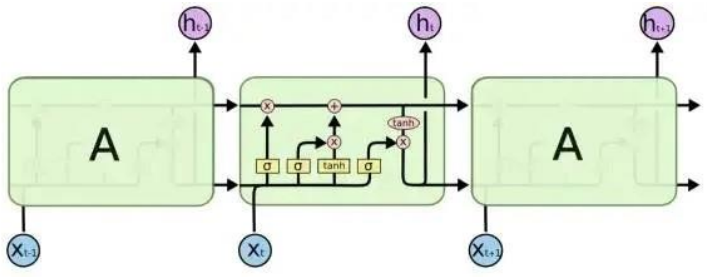


Fig. 4. LSTM structure

3 Vulnerable code dataset

Due to the recent rise of using deep learning for defect detection research in the past few years, there is currently no unified and accepted dataset for evaluating defect detection models in contrast to some classical and mature de ep learning tasks. Currently, researchers often have to construct their own def ect code datasets as a basis for evaluating model performance. This leads to t he usage of different self-built datasets in various studies. However, comparin g different models based on numerical metrics on different datasets is not real istic. Therefore, the comparison between models is often avoided in this field of research. So far, no research has specifically pointed out the issues cause d by the lack of dataset standardization. Additionally, the process of constructi ng self-built defect code datasets also suffers from ambiguity or approximation s, and even involves some unreasonable operations, which can significantly im pact the training performance of models.

Unlike tasks such as image processing and natural language processing th at have mature and widely-used datasets, using deep learning for defect detect ion faces significant challenges in terms of dataset availability. There are two core difficulties at the heart of these challenges:

- The number of defect samples in known real-world projects is limited.
- The difficulty and cost of manual annotation or automated generation are extremely high.

3.1 Classification of construction methods of defect code data s et

The process of constructing defect code data set can be divided into three main technical links: the acquisition of defect items, the extraction and proce ssing of defect codes, and the labeling of processed samples. Therefore, corres ponding to each technical link, we classify it according to data source, sampl e granularity and label source, as shown in Figure 5.

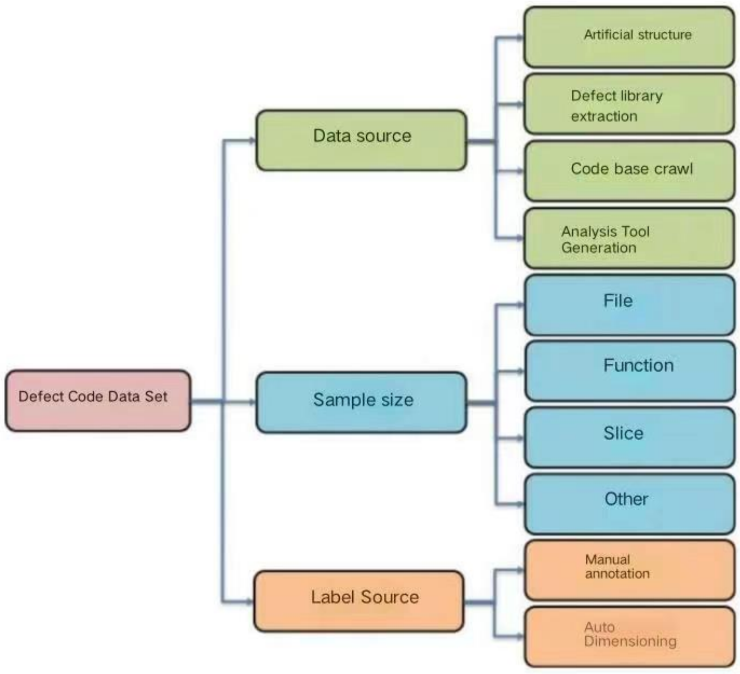


Fig. 5. Classification of construction methods of defect code dataset

Constructing a dataset of defective code requires acquiring defect data by finding code related to defects. Based on the data source, there are currently four main types of defect data sources: manual creation, extraction from defec t databases, crawling code repositories, and generating using analysis tools:

- Manual creation involves transforming correct code into defective cod e using manual rules.
- Extraction from defect databases involves extracting entries of specifie d defect types from publicly available defect databases and extracting corresponding defective code.
- Crawling code repositories involves directly searching code repositorie s and scraping submitted code with defective features.
- Generating using analysis tools involves using defect analysis tools to detect defects in source code and using the results as labels for whe ther they contain defects.

Once it is confirmed that defects exist in the code, selecting an appropria te granularity becomes a critical issue. If the granularity is too large, it could introduce more noise and redundancy. If it's too small, it might result in the loss of defect characteristics. Moreover, different granularities require differen t network structures. Good granularity combined with a suitable network can help the model better capture defect features. Currently, the main sample gran ularities are file-level, function-level, slice-level, and other categories:

- File-level mainly involves splitting projects into file-level to form a d ataset.
- Function-level mostly obtains file data first and then divides it into in dividual samples based on functions.
- Slice-level requires parsing the code and slicing it according to predet ermined rules.

After obtaining processed samples, they need to be labeled for whether th ey contain defects:

- If the defective samples are generated using manual rules, they alread y have labels, which can be automatically annotated.
- When extracting defect data from defect databases if adopted using th e same granularity as the data source, the labels are often already obt ained, and they can be automatically annotated.

- For finer granularity defect labeling, researchers need to perform man ual labeling work themselves. Currently, the main methods of obtainin g labels are automatic annotation and manual annotation. For ease of explanation, we use the data source as a classification dimension to i ntroduce the current methods for constructing defect code datasets.

3.2 Comparison of data set construction methods

As shown in Table I, the representative data set construction methods in r ecent years are listed according to the technical route of each technical link, and the advantages and disadvantages of each technical route are evaluated.

From the point of view of data source, different data sources mean differ ent sample complexity. For example, NVD extraction data widely used in def ect library extraction and Sard test set data widely used in artificial constructi on: the former comes from real industrial code, which is highly complex and modular, and the dependency and call relationship between components are c omplex; The latter comes from artificially written test cases, and its complexit y is low, and it mainly focuses on the expression of defects, but mostly igno res its real function. Therefore, it is an independent program fragment, and th ere is almost no long-range dependence and complex calling relationship betw een components. From the perspective of practical application, the defect dete ction system needs to have the ability to detect real industrial codes. That is to say, the ideal defect code data set should come from industrial codes such as NVD instead of manually written test cases, but the difficulty and cost of obtaining industrial code defect data are much greater than that of test cases, which requires more perfect sample extraction technology and a lot of inevit able manual review.

From the point of data granularity, there is a gap in completeness and re dundancy between different data granularities. File-level data can contain more paths, but it inevitably brings a lot of irrelevant noise. Function-level data is extremely low in cost when it is divided, and it is not necessary to analyze the tested code (just cut it into functions) in the detection process. However, Function-level data is based on the naive assumption that defects occur withi n the scope of functions. This is obviously difficult to support in large-scale i ndustrial code, especially in the case of modularity. The declaration and use o f data often do not exist in a single function. Slicing-level data can support c ross-functional defect paths, but slicing operations need to analyze the data flo w and control flow of the code when building data sets and detecting the co de under test, which requires a lot of resources and time costs.

In terms of labeling methods, different labeling methods make the accurac y of labels different from the acquisition cost. Manual labeling by security-rel ated personnel can ensure the quality of labels, but its acquisition cost is hig h and the labeling speed is slow, so it takes a long-term accumulation to reac h the data scale required by deep learning model. Automated labeling with he uristic method can generate labels quickly and in large quantities, but its label ing accuracy is difficult to achieve the ideal labeling effect.

Table I Summary of data set construction methods

Technical l ink	Technology ro admap	document	advantage	disadvantage
data source	Artificial struc ture	[5,6]	Low cost and large quantity.	The construction rules are simple and the complexity is low.
	Defect library extraction	[7,8] [9,10]	High complexity, close to industrial sc ene, and manual confirmation.	The cost is high and the num ber of excavations is small.
	Code base cra wling		High complexity, close to the industria l scene	High cost, lack of defect type s
	Analysis tool generation	[11]	Low cost and large quantity.	Limited by tool accuracy
Sample gra nularity	document	[6,12]	Natural division, low cost	Coarse granularity and mo re redundant information.
	function	[5,9]	Natural division, low cost	Cross-functional defects are not su pported, and there are many redun dant information.
	slice up	[7,8]	Support cross-functional defects, less r edundant information	Computational resources are expe nsive and slow.
	other	[13]		
Label sour ce	Manual marki ng	[10]	High labeling accuracy	The cost is extremely high and the speed is slow
	Automatic lab eling	[7,8]	Low cost and high speed.	Low labeling accuracy

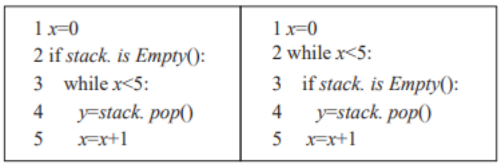
4 Vulnerability detection model based on deep le arning

4.1 Vulnerability detection based on software metrics

The vulnerability prediction model based on software metrics represents in formation related to software vulnerabilities by artificially designing software f eatures, and then uses machine learning algorithms to predict and identify co mponents related to vulnerabilities. The vulnerability prediction model uses me trics such as complexity, code change, developer activity, coupling and cohesi on, and uses machine learning algorithms such as LR, SVM, J48, DT, RF, N B and BN to predict components related to vulnerabilities.

Literature [14] makes a case study of JavaScript engine in Mozilla, and fi nds that nine complexity metrics (including McCabe, SLOC, etc.) can be used to predict vulnerabilities, but the false positive rate is high. The author of lit erature [14] finds that the complexity metrics of vulnerability function and def ect function are only significantly different in complexity. Therefore, complexit y can distinguish vulnerabilities from defect functions. Literature [15] uses me trics such as source code lines and code changes to predict vulnerabilities, an d the results show that the regression tree model achieves 100% recall rate a nd 8% false positive rate in the best case. Literature [16] studies the relations hip between developer collaboration structure and vulnerabilities, and it is fou nd that, Files changed by 9 or more developers are 16 times more likely to have vulnerabilities than files changed by less than 9 developers. Literature [1 7] studies whether three software metrics (complexity, code change and develo per activity) can distinguish vulnerable files to guide security inspection and t esting. The results show that, At least 24 of the 28 metrics can distinguish v ulnerable files and neutral files of two projects, and these three metrics can p rovide valuable guidance for security inspection and testing. Reference [18] de veloped a vulnerability prediction framework based on complexity, coupling an d cohesion, which can correctly predict most files affected by vulnerabilities i n Mozilla Firefox with a low false positive rate.

The research on vulnerability prediction based on software metrics puts fo rward a large number of indicators, trying to manually extract vulnerability-rel ated features from software code as completely as possible. However, these ar tificially defined features may not fully understand the meaning of the code it self, so there are still many shortcomings in actual vulnerability prediction. Fo r example, the two Python codes in Figure 6 all pop up five data from the t op of the stack in terms of function. However, when there are less than five data in the stack, there is a risk of code 1 spilling down, while code 2 does not. However, these two pieces of code have the same software metrics, cod e tags and frequencies, and it is difficult to distinguish them only by manuall y extracted indicators. Therefore, an algorithm that can extract the grammatica l structure features and semantic information content of the code is needed.



```
x=0
if stack. is Empty():
  while x<5:
    y=stack. pop()
    x=x+1
```

```
x=0
while x<5:
  if stack. is Empty():
    y=stack. pop()
  x=x+1
```

Fig. 6. Code example

4.2 Vulnerability detection based on grammar and semantics

Grammatical-semantic-based vulnerability mining extracts feature informatio n from software code through text mining, AST, etc., and transforms it into d igital representation of vectors, and then uses machine learning (ML) or deep learning (DL) to classify it. Graph algorithm or attention mechanism can also be used to specifically locate possible vulnerabilities. The research based on grammatical semantics mainly focuses on exploring different feature extraction methods and various training models.

Literature [19] proposes a method that relies on source code text analysis, and converts each file into a feature vector. This method regards each letter combination of source code as a feature. Using the counting value of the giv en letter combination in the source code of a given file, this research has ach ieved 87% accuracy, 85% accuracy and 88% recall on the open source mobil e application. Literature [20] takes the code mark and frequency in the file as the text features of the code to predict the components that may have vulner abilities. Compared with the software metric model of PHP application, this m odel achieves a higher recall rate. However, for the word bag model, it ignor es the syntactic structure information between codes, and the semantic informa tion hidden in the program can help vulnerable codes provide richer represent ation, thus improving the effect of the vulnerability prediction model. Later re search introduced AST to characterize the syntactic structure. Literature [21] e xtracts AST from codes and determines the structural pattern of trees, and aut omatically compares codes according to the structural pattern of trees. This m ethod can predict vulnerabilities only by checking a small number of code ba ses. Literature [22] uses Antlr to extract AST from C/C++ source files, extrac ts vectors from AST with each function as the basic unit, and uses this vecto r to train classifiers to predict buffer vulnerabilities.

Many deep learning techniques have also been applied to the task of vuln erability prediction. Deep learning techniques can automatically obtain deeper f eature information from complex codes to characterize the syntax and semanti c features of programs. Literature [23] uses deep neural network, combined w ith N-gram analysis and feature selection to construct features on token-level data. This model can achieve high precision, high accuracy and high recall in Java Android application vulnerability prediction. Literature [24] uses LSTM to capture the context in the source code and learn the syntax and semantic f eatures of the code. The results show that this method is superior to the tradi tional software measurement model. Xian studied the value of word embeddin g algorithm in vulnerability prediction based on text mining. Word2vec and fa st-text models are used to learn the syntax and semantic relationship between code tags, and CNN and RNN models are used to predict whether there are l oopholes in files. The author of reference found that the F2-score of the mod el increases when the algorithm used to generate word embedding vectors is used, and the word2vec algorithm has the best effect. The vulnerability predic tion model based on deep learning has also become a research hotspot.

Document proposed the VulDeePecker system, which introduced the deep neural network BiLSTM into the vulnerability prediction task, and collected a data set for evaluating the vulnerability detection effect. Document [25] estab lished a vulnerability prediction system based on deep learning, which can ma ke coarse-grained prediction for input files or functions, and also for a small code block in a new example. Fine-grained prediction and bit estimation of s ome vulnerabilities are carried out. Literature [26] puts forward Devign model based on GNN, and extracts four attributes AST, CFG, data flow graph, DF G) and the code sequence in natural language to complete the representation of the code. The SySeVR model proposed by reference [27] uses semantic in formation based on data dependency to represent the code. The IVDetect mod el proposed by reference extracts the data in the code and controls the depen dency to generate a vector representation of five dimensions, which is unifor mly trained and learned by RNN model. The experimental results show that, The IVDetect model improves the accuracy and can locate the function block where the bug is located. Based on this, the reference proposes the LineVul model, which uses the CodeBERT pre-training model to complete the word ve ctor transformation of the code, uses the line granularity data set for training, and uses GNN to capture the dependency in the text. The model can get hi gher prediction accuracy and locate the code line where the bug is located.

Extracting the vulnerability features in files from the perspective of text mining can more fully capture the syntax and semantic features of codes. The improvement of computer computing power and the introduction of deep lear ning methods also enable researchers to extract rich features from complex an d lengthy codes. Scholars in this field mainly focus on how to complete the transformation from codes to continuous value vectors and build deep learning models to optimize and improve vulnerability prediction tasks. Through sum mary, it is found that the code features are In the stage of characterization, A ST and RNN are chosen by most researchers. AST can help to understand th e logical structure in code, while RNN can capture long-distance dependencies in files with huge code. In addition to AST and RNN, the work in recent y ears has gradually turned to large-scale language models (such as CodeBERT). These large-scale language models have been pre-trained on programming lan guage tasks, so they can better capture and understand programming language features and apply them to various downstream tasks related to programming languages.

5 Summary and prospect

This section summarizes the challenges and opportunities faced by these t wo research fields at this stage by combing and summarizing the research res ults of vulnerability detection tasks, as shown in Table II.

Table II Opportunities and Challenges of Vulnerability Detection Task

challenge	opportunity
Source and processing of data se t	Establish a high-quality balanced and noise-free benchmark data set.
Representation method of code v ector	Construct a representation method that contains grammatical and semanti c information to the greatest extent.
Improvement of pre-training mod el	Improve the performance of the model by embedding word vectors train ed in other fields.
Exploration of deep learning mo del	Explore a deep learning model that is more suitable for specific forecasti ng tasks.
Fine-grained prediction technolog y	More accurately locate the possible locations of defects and vulnerabilitie s.
Migration of pre-training model	Save time and resource costs through model migration.

Software vulnerability detection can reduce the cost of software repair and improve the quality of products by means of machine learning or deep learn ing. Studying open source software vulnerability detection can improve the un derstanding of software defects in various projects and better guide code detec tion and testing. This paper investigates and analyzes the relevant literature in the field of software prediction research, taking machine learning and deep l earning as the starting point. Two kinds of prediction models based on softwa re metrics and syntax semantics are sorted out. Finally, six hot issues of soft ware prediction are analyzed in detail, and the future development direction of software vulnerability detection is pointed out. By summarizing the research related to software defect prediction, this paper holds that the future research direction can be carried out from four aspects:

- creating a high-quality defect data set is helpful to predict defects, im prove resource allocation and repair defective codes;
- Construct a grammar with maximum implication.

The representation method of semantic information and word vector embeddin g trained in other fields can improve the performance of defect prediction mo del;

- Adapting to fine-grained prediction technology can locate the possible positions of defects and vulnerabilities more accurately;
- Developing a general model to distinguish defects from vulnerabilities can better predict the vulnerable code locations.

reference

[1] Common Vulnerabilities & Exposures. Published CVE Records. [Online] Available: h ttps://www.cve.org/About/Metrics.

[2] Planning S. The economic impacts of inadequate infrastructure for software testing. Technical Report, National Institute of Standards and Technology, 2002.

[3] Hindle A, Barr ET, Gabel M, et al. On the naturalness of software. Communications of the ACM, 2016, 59(5): 122−131.

[4] Perl H, Dechand S, Smith M, et al. VCCfinder: Finding potential vulnerabilities in o pen-source projects to assist code audits. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. 2015. 426−437.

[5] Duan X, Wu J, Ji S, et al. VulSniper: Focus your attention to shoot fine-grained vu lnerabilitie. In: Proc. of the IJCAI. 2019. 4665−4671.

[6] Ghaffarian SM, Shahriari HR. Neural software vulnerability analysis using rich inter mediate graph representations of programs. Information Sciences, 2021, 553: 189−20 7.

[7] Li Z, Zou D, Xu S, et al. VulDeePecker: A deep learning-based system for vulnerab ility detection. NDSS, 2018.

[8] Zou D, Wang S, Xu S, et al. μVulDeePecker: A deep learning-based system for mul ticlass vulnerability detection. IEEE Trans. on Dependable and Secure Computing, 20 19.

[9] Li Y, Wang S, Nguyen TN, et al. Improving bug detection via context-based code r epresentation learning and attention-based neural networks. Proc. of the ACM on Pro gramming Languages, 2019, 3(OOPSLA): 1−30.

[10] Zhou Y, Sharma A. Automated identification of security issues from commit messa ges and bug reports. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. 2017. 914−919.

[11] Russell R, Kim L, Hamilton L, et al. Automated vulnerability detection in source c ode using deep representation learning. In: Proc. of the 17th IEEE Int'l Conf. on M achine Learning and Applications (ICMLA). IEEE, 2018. 757−762.

[12] Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural networ k. In: Proc. of the 2017 IEEE Int'l Conf. on Software Quality, Reliability and Secur ity (QRS). IEEE, 2017. 318−328.

[13] Pradel M, Sen K. Deepbugs: A learning approach to name-based bug detection. Pro c. of the ACM on Programming Languages, 2018, 2(OOPSLA): 1−25.

[14] Gegick M, Williams L, Osborne J, et al. Prioritizing software security fortification through code-level security metrics[C] Proc of Workshop on Quality of Protectio n. New York: ACM, 2008: 31−38

[15] Meneely A, Williams L. Secure open source collaboration: An empirical st udy of Linus' law[C] Proc of the 16th ACM Conf on Computer and Co mmunications Security. New York: ACM, 2009: 453−462

[16] Shin Y, Meneely A, Williams L, et al. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnera bilities[J]. IEEE Transactions on Software Engineering,2010, 37（6）: 772−78

[17]Chowdhury I, Zulkernine M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities[J]. Journal of Systems Archite cture, 2011, 57（3）: 294−31

[18]Hovsepyan A, Scandariato R, Joosen W, et al. Software vulnerability pred iction using text analysis techniques[C]//Proc of the 4th Int Work shop on Security Measurements and Metrics. New York: ACM, 2012: 7−1 0

[19]Scandariato R, Walden J, Hovsepyan A, et al. Predicting vulnerable softw are components via text mining[J]. IEEE Transactions on Software Engineering, 2014, 40（10）: 993−1006

[20]Yamaguchi F, Lottmann M, Rieck K. Generalized vulnerability ext rapolation using abstract syntax trees[C]//Proc of the 28th Annual

[21]Computer Security Applications Conf. New York: ACM, 2012: 35 9−368 Meng Qingkun, Wen Shameng, Feng Chao, et al. Predicti ng buffer overflow using semi-supervised learning[C]//Proc of 2016 9th Int Congress on Image and Signal Processing, BioMedical Enginee ring and Informatics (CISP-BMEI). Piscataway, NJ: IEEE, 2016: 19 59−1963

[22]ang Yulei, Xue Xiaozhen, Wang Huaying. Predicting vulnerable so ftware components through deep neural network[C]//Proc of the 2 017 Int Conf on Deep Learning Technologies. New York: ACM, 2017: 6−10

[23]Dam H K, Tran T, Pham T, et al. Automatic feature learning for predicting vulnerable software components[J]. IEEE Transaction s on Software Engineering, 2018, 47（1）: 67−85

[24]Kalouptsoglou I, Siavvas M, Kehagias D, et al. An empirical e valuation of the usefulness of word embedding techniques indeep learning-based vulnerability prediction[C]//Proc of Int ISCIS Securit y Workshop. Berlin: Springer, 2022: 23−37

[25]Li Zhen, Zou Deqing, Xu Shouhuai, et al. SySeVR: A framework for usi ng deep learning to detect software vulnerabilities[J]. IEEE Transa ctions on Dependable and Secure Computing, 2021, 19（4）: 2244 −2258

[26]Li Yi, Wang Shouhua, Nguyen T N. Vulnerability detection with fine grai ned interpretations[C] Proc of the 29th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software En gineering. New York: ACM, 2021: 292−303

[27]Fu M, Tantithamthavorn C. LineVul: A transformer-based line-levelvulnera bility prediction[C] Proc of 2022 IEEE/ACM 19th Int Confon Mining S oftware Repositories (MSR). Piscataway, NJ: IEEE,2022: 608−620