This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3034766, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.20xx.DOI

Software Vulnerability Analysis and Discovery using Deep Learning Techniques: A Survey

PENG ZENG[1], GUANJUN LIN[2], LEI PAN[3],YONGHANG TAI[1] AND JUN ZHANG[1],

[1]School of Physics and Electronic Information, Yunnan Normal University, Kunming, 650000, China
[2]School of Information Engineering, Sanming University, Sanming, Fujian, 365004, China
[3]School of Information Technology, Deakin University, Geelong, VIC, 3220, Australia

Corresponding authors: Jun Zhang and Yonghang Tai. Peng Zeng and Guanjun Lin contribute equally to this paper.

ABSTRACT Exploitable vulnerabilities in software have attracted tremendous attention in recent years because of their potentially high severity impact on computer security and information safety. Many vulnerability detection methods have been proposed to aid code inspection. Among these methods, there is a line of studies that apply machine learning techniques and achieve promising results. This paper reviews 22 recent studies that adopt deep learning to detect vulnerabilities, aiming to show how they utilize state-of-the-art neural techniques to capture possible vulnerable code patterns. Among reviewed studies, we identify four game changers that significantly impact the domain of deep learning-based vulnerability detection and provide detailed reviews of the insights, ideas, and concepts that the game changers have brought to this field of interest. Based on the four identified game changers, we review the remaining studies, presenting their approaches and solutions which either build on or extend the game changers, and sharing our views on the future research trends. We also highlight the challenges faced in this field and discuss potential research directions. We hope to motivate the readers to conduct further research in this developing but fast-growing field.

INDEX TERMS deep learning, vulnerability detection.

I. INTRODUCTION

INCREASINGLY more cyberattacks are rooted in software vulnerabilities, resulting in the leak of user data and the damage of the company's reputation [33], [37]. Although many studies have been proposed to aid vulnerability detection, vulnerabilities remain threats to the secure operation of IT infrastructure [74]. The number of vulnerabilities disclosed in the Common Vulnerabilities and Exposures (CVE) [1] and the National Vulnerability Database (NVD) [2] representing the vulnerability data repositories increased from approximately 4,600 in 2010 to 8,000 in 2014 before jumped to over 17,000 in 2017 [32], [63]. These vulnerabilities may have posed potential threats to the secure usage of digital products and devices worldwide [6], [7], [40], [73].

Aiming to identify vulnerabilities before the deployment of software, many vulnerability detection methods have been proposed to combat the attacks caused by vulnerability exploitation [31], [32], [36], [37], [92]. In addition to the software community's detection solutions, many studies advocate conventional machine learning techniques [2], [65], [68], [69], [87], [88]. However, the conventional machine learning-based solutions usually require experts to explicitly define features [46], [65], [66]. In many cases, the manually defined features can be task-specific, subjective, and error-prone [83], [84]. Furthermore, the quality of the manually determined features is inherently confined by practitioners' experience and knowledge. Deep learning techniques extract features automatically, resulting in relieving experts from tedious feature engineering tasks. The abstract feature representations automatically extracted by deep learning methods often demonstrate better generalization abilities than manually extracted features [35]. These benefits accelerate the adoption of deep learning techniques in newly proposed software vulnerability detection solutions [9], [11], [12], [32], [37].

This paper reviews 22 recently published studies that apply deep learning for vulnerability detection. The focus is on how

[1]https://cve.mitre.org
[2]https://nvd.nist.gov

to use the emerging neural network techniques for capturing potentially vulnerable code patterns. Among the reviewed literature, we identify and prioritize four revolutionary studies as `game changers` according to the significance of their impacts on deep learning-based vulnerability detection. For the four game changers, we review their technical details, innovative insights, and significant impacts. We categorize the remaining 18 studies into different categories relevant to the four game changers. We exploit the game changers and their follow-up studies to elaborate on the research problems and the solutions with a systematic view. Finally, we share our views on the challenges in this field and point to potential research directions.

Related Reviews

Several surveys summarize various solutions for applying machine learning and deep learning for bug or vulnerability detection. Liu et al. [38] briefly reviewed the vulnerability detection solutions using code analysis and machine learning techniques. Malhotra et al. [45] reviewed software fault prediction solutions using machine learning techniques. Radjenovic et al. [54] provided a systematic literature review to depict the state-of-the-art software metrics in software fault prediction. However, fault prediction is marginally relevant to vulnerability detection [74]. Ghaffarian and Shahriari [15] presented an extensive and in-depth review, summarizing studies applying traditional machine learning techniques for vulnerability detection, excluding deep learning methods. Wang et al. [77] reviewed machine learning-based fuzzing techniques for vulnerability discovery. Ji et al. [24] briefly reviewed the studies of adopting automated systems for detecting, patching, and exploiting software vulnerabilities.

However, only a few surveys investigate deep learning-based solutions for vulnerability detection. Singh et al. [70] provided a brief survey of methods utilizing deep learning-based software vulnerability discovery with feasibility analysis. Lin et al. [33] examined studies focusing on how deep learning techniques facilitate the understanding of code semantics for vulnerable pattern recognition.

Contributions of this Survey

In this survey, we use a different way to review the state-of-the-art research papers using deep learning for vulnerability detection. We identify four “game changers" which we consider as milestones in the field. Then, we categorize the reviewed studies based on their relatedness to the game changers. When reviewing each study, we examine whether the work follows the trend led by a specific game changer, aiming to provide a different perspective to see the approaches proposed in applying deep learning for vulnerability detection. By reviewing the game changers and their subordinates, we highlight trends and the challenges faced in the field of interest, showing the status and achievements of this cutting-edge research.

Paper Organization

The paper is organized as follows: Among 22 reviewed papers, we identify four papers as game changers and explain our choice in Section II. By highlighting the influence that the game changers have exerted on the filed of deep learning-based vulnerability detection, we discuss the main problems faced and the solutions proposed. In Section III, we review the remaining papers and categorize them based on their relevance to the game changers, focusing on the research problems, methods, and solutions they proposed. Section IV elaborates on the research challenges and possible future directions in the field of vulnerability detection. Section V concludes this paper.

II. GAME CHANGERS LEADING THE RESEARCH TREND

Applying deep learning for vulnerability detection is an emerging field. During the review process, we identify four research works that have laid a foundation for deep learning-based vulnerability detection. Many remaining studies are followed and built based on the four papers. Therefore, to better understand the trends in this field, in this section, we firstly identify these game changers with highlights of concepts, ideas, and deep learning models. Because of their unique and significant contributions, we clearly articulate how the subordinate works further explore the field.

A. IDENTIFYING GAME CHANGERS

The four game changers which have a significant impact are as follows:

- **Game changer 1**: “Automatically Learning Semantic Features for Defect Prediction" by Wang et al. [75]. It is the first paper that proposes to use a deep learning algorithm for learning the *semantics code representations* indicative of defective code. Wang et al. [75] advocated that semantic information in the program’s source code should be extracted through the deep learning algorithms’ representation learning capability. It means that the feature sets characterizing defective code can be automated without relying on manual extraction. Hence, feature representations are automatically extracted by deep learning, relieving experts from tedious and time-consuming feature extraction processes.
- **Game changer 2**: “End-to-End Prediction of Buffer Overruns from Raw Source Code via Neural Memory Networks" by Choi et al. [4]. It is the first paper to provide an *end-to-end* solution for vulnerability detection, which means that programs’ source code can be direct inputs to a competent neural model, which is the *memory networks* [78] and the output is whether the corresponding inputs being vulnerable or not. The end-to-end solution demonstrates that it is feasible for a neural model to directly operate on source code, without requiring any code analysis for pre-processing, which significantly automates the vulnerability detection process.

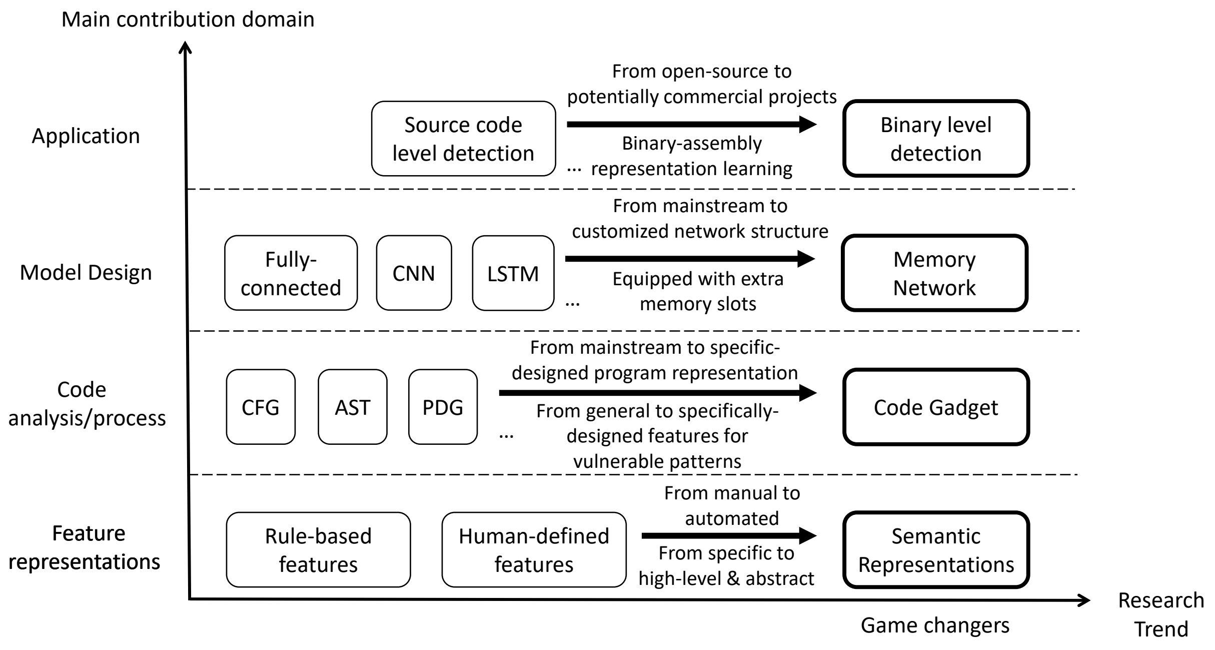

Main contribution domain
Application
Model Design
Code analysis/process
Feature representations
Source code level detection
From open-source to potentially commercial projects
Binary-assembly representation learning
...
Binary level detection
Fully-connected
CNN
LSTM
From mainstream to customized network structure
Equipped with extra memory slots
...
Memory Network
CFG
AST
PDG
From mainstream to specific-designed program representation
From general to specifically-designed features for vulnerable patterns
...
Code Gadget
Rule-based features
Human-defined features
From manual to automated
From specific to high-level & abstract
Semantic Representations
Game changers
Research Trend

FIGURE 1. The contribution domains from the identified game changers.

- **Game changer 3**: “VulDeePecker: A Deep Learning-Based System for Vulnerability Detection" by Li et al. [32]. This paper proposes the “*code gadgets*" concept consisting of multiple lines of code depicting data dependencies or control dependencies. Compared with existing studies that use abstract syntax trees (ASTs) for learning potentially vulnerable patterns [36], [37], the code gadget is a more fine-grained entity for revealing variable flaws. This setup allows a neural model to obtain accurate and precise information relevant to specific vulnerability types, e.g., the buffer error and resources management error vulnerabilities. This paper is a pioneer that applies fine-grained program representation for the neural network to learn high-level representations.
- **Game changer 4**: “Cyber Vulnerability Intelligence for Internet of Things Binary" by Liu et al. [41]. This work extends the application domain of neural model-based vulnerability detection from source code to *binary* code. In practice, commercial software and the firmware of the Internet of Things (IoT) devices are provided in binary code. This work enables vulnerability detection to be performed without relying on the source code.

Deep learning-based vulnerability detection methods usually employ four steps, including data collection, data preparation, model building, and evaluation/test. Data collection is to gather labeled vulnerable and non-vulnerable data for neural model training. Data preparation aims to convert raw data (in most cases, the data in textual format) to vector representations acceptable by neural models. These two steps are related to the data, which is the key to building an effective detection system. Model building is to apply or customize a deep neural network model to extract potentially vulnerable patterns for building a vulnerability detector. Evaluation/test evaluates or tests the built detector in specific application scenarios. Therefore, the four game changers have made contributions in different domains, as shown in Figure 1.

Figure 1 shows the key contribution domains made by the identified game changers. Game changer 1 [75] has revolutionized code feature learning because high-level and abstract features indicative of defective code can be automatically extracted by a neural network, instead of manual feature engineering. Game changer 2 [4] forsakes mainstream program representations such as Control Flow Graph (CFG), AST, and Program Dependency Graph (PDG) for learning vulnerable code features. Instead, a novel program representation called “code gadget" was designed to reveal vulnerable programming patterns and facilitate the neural network to learn vulnerability-relevant features. Game changer 3 [32] is the first work to apply a customized network structure called memory networks for building a detection model. Compared with mainstream network structures like fully-connected network, CNN, and LSTM, the memory networks use extra built-in memory slots for memorizing long-range code dependencies that are crucial for identifying buffer error vulnerabilities. Game changer 4 [41] has extended the application domain of deep learning-based vulnerability detection from source code to binary code.

An extensive review of all four game changers is provided in the following subsections, detailing how they contribute to the field of interest.

B. GAME CHANGER 1 – "AUTOMATICALLY LEARNING SEMANTIC FEATURES FOR DEFECT PREDICTION"

Gamer changer 1 [75] is the pioneer study to adopt a deep belief network (DBN) to learn the semantic representations of a program. It aims to use the high-level semantic representations learned by the neural networks as defective features. In particular, it enables the automated learning of feature sets indicative of defective code, without relying on manual feature engineering. This method is not only suitable for within-project defect prediction but also applicable for cross-project defect prediction. AST is used as the representation of the program to feed into a DBN for data training. A data processing method is proposed with four steps. The first step is to parse the source code into a token; the second step is to map the token to an integer identifier; the third step is to use DBN to generate semantic features automatically; the final step is to use DBN is to establish a defect prediction model.

Experiments were conducted on open-source Java projects dataset to assess the performance of this proposed method. The empirical studies demonstrate that the method can effectively and automatically learn semantic features from Within-Project Defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP). However, this paper's detection granularity is not fine-grained enough to allow code inspectors to pinpoint the vulnerabilities related to specific code lines because it only works at the file level. The DBN used for learning semantic representations has motivated follow-up researchers to apply various types of neural networks for learning abstract feature representations for defect and vulnerability detection, such as [29], [35]–[37], [42], [47], [59]. Hence, expert-defined features are not the necessity when deep learning is involved. Thus experts can devote their effort to tailoring code analysis, neural model design, or both.

C. GAME CHANGER 2 – "END-TO-END PREDICTION OF BUFFER OVERRUNS FROM RAW SOURCE CODE VIA NEURAL MEMORY NETWORKS"

Game changer 2 [4] is the first to provide an end-to-end solution for detecting buffer error vulnerabilities. Empirical studies demonstrate that a neural network is capable and expressive enough to directly learn vulnerability-relevant features from raw source code without code analysis. A novel neural model is developed to relax the code analysis constraint by constructing and customizing the memory networks [72], [79]. The proposed neural network equips with built-in memory blocks to memorize very long-range code dependencies. Hence, this network modification is crucial for identifying buffer error vulnerabilities. For performance evaluation, experiments are conducted on a self-generated dataset. Experimental results show that this method can accurately detect different types of buffer overflows. In terms of solving overflow tasks, the memory networks model is superior to other models. However, this method still has limitations for further improvements. The first limitation is that it fails to detect the buffer overflow issues residing in external functions because the predefined code in external files is excluded in the input data. The second limitation is that each row must contain some data assignment for this model to work. It is not easy to apply this method out of the box for the source code containing conditional statements because attention values are calculated to find the most relevant code positions.

This paper uses source code as the input of a memory network architecture to train the model directly, has motivated follow-up researchers to apply an end-to-end solution to use neural networks to train source code data directly [34], [64]. It mitigates the code analysis constraint so that the neural network can gain an in-depth understanding without excessive code analysis.

D. GAME CHANGER 3 – "VULDEEPECKER: A DEEP LEARNING-BASED SYSTEM FOR VULNERABILITY DETECTION"

Game changer 3 [32] is the first study to apply a bidirectional Long Short Term Memory (BiLSTM) model [20] for vulnerability detection. The BiLSTM extracts and learns long-range dependencies from code sequences. Its training data is derived from the code gadget representing the program fed to the BiLSTM. There are three stages to process the code gadgets. The first stage is to extract the corresponding program slices of library/API function calls. The second stage is to generate and label code gadgets. The third stage is to convert the code gadgets into vectors. After that, the dependencies across a long-range are captured from the code gadgets. In this paper, the granularity of detection is at the slice-level. For evaluating the performance of VulDeePecker, a set of experiments is conducted on open-source projects and the SARD dataset [48]. The experimental results show that VulDeePecker can handle multiple vulnerabilities, and human experience can help improve the effectiveness of VulDeePecker. Besides, VulDeePecker is more effective than other static analysis tools that require experts' defined rules for detecting vulnerabilities. However, this method has limitations for further improvement. The first limitation is that VulDeePecker only can deal with C/C++ programs. The second limitation is that VulDeePecker can only deal with vulnerabilities related to library/API function calls. The third limitation is that the dataset for performance evaluation is small-scale as it only contains two vulnerability types.

The BiLSTM model used in VulDeePecker has motivated researches to adopt this paradigm. The use of BiLSTM enables research works [28], [43] to inspect the code dependencies across long-range. VulDeePecker proposes to use fine-grained code gadgets as the representation of a program instead of capturing data dependence relations only. This paper also inspired an extended work in [92], where code gadgets represent programs to capture the data dependence relation and control dependence relation. Besides, this paper and a few extended works [30], [31] come from the same research team with continuous improvements and innovations.

E. GAME CHANGER 4 – "CYBER VULNERABILITY INTELLIGENCE FOR INTERNET OF THINGS BINARY"

Game changer 4 [41] proposes a deep learning-based vulnerability detector for binary code. It aims to broaden the application field of vulnerability detection by mitigating the unavailability of source code. For data training, binary segments are fed to a bidirectional LSTM neural network with attention (Att-BiLSTM). The data processing consists of three steps. First, the binary segments were obtained by applying the IDA Pro tool on the original binary code. The second step extracts functions from binary segments and mark them as 'vulnerable' or 'not vulnerable.' The third step takes the binary segment as a binary feature before feeding it to the embedding layer of the Att-BiLSTM. Moreover, the granularity of detection is at the function-level. For evaluating the performance of the proposed method, many experiments are conducted on an open-source project dataset. The experiment results indicate that the proposed method outperforms the source code-based vulnerability detection approaches on binary code. However, this method has limitations for further improvement. In particular, the detection accuracy is relatively low, given that the detection accuracy is less than 80% in each dataset. It also overlooks the function inlining scenario when the binary code's structure often changes due to function inlining.

This paper utilized deep learning techniques to detect vulnerabilities for binary code. It has motivated follow-up works to detect binary code vulnerability with deep learning methods [27], [82]. It paves a path to expand deep learning-based vulnerability detection applications from source code to binary code.

III. TRENDS FOR DEEP LEARNING-BASED VULNERABILITY DETECTION

A. CATEGORIZATION BASED ON GAME CHANGERS

The game changers are trend-leaders to inspire multiple follow-up papers. In this paper, we categorize the review studies into five categories corresponding to the four game changers:

1) **Methods based on semantic representation learning:** Game changer 1 is the first study that applies neural networks for learning code semantic representation indicative of potentially vulnerable code semantics. The follow-up studies [29], [35]–[37], [42], [47], [59] inherit the similar idea. Hence, we categorize them into one category.
2) **Solutions capable of end-to-end detection:** Game changer 2 proposes an end-to-end detection solution, allowing the source code to be directly fed as inputs to a neural network. The follow-up papers [34], [64] of game changer 2 belong to this category.
3) **Features designed for vulnerable patterns:** Game changer 3 utilizes specifically designed features to facilitate the neural models to learn potentially vulnerable patterns. The follow-up studies [28], [30], [31], [43], [92] extend game changer 3, thus belonging to this category.
4) **Vulnerability detection for binary code:** Based on game changer 4, the follow-up studies [27], [82] investigate vulnerability detection for binary code.
5) **Miscellaneous:** We assign the remaining works [18], [49] to the miscellaneous category because they do not belong to any of the aforementioned categories.

The following subsections present the review of each study, aiming to identify the research problems, the methodologies, and the effectiveness of the proposed solutions.

B. FOLLOW-UPS OF GAME CHANGER 1

This subsection reviews seven follow-up studies of game changer 1, presenting the problem identified, approaches and solutions proposed for vulnerability discovery. The key difference of reviewed studies is listed in Table 1 from the perspectives of data source, features, neural network models, and detection granularity.

A deep belief network (DBN) is used in game changer 1 for extracting semantic features from abstract syntax trees (ASTs) built from Java source code. However, the original detection is performed at the file-level. As an improvement, DeepBalance [42] applies BiLSTM networks to learn the code representations at the function-level to pinpoint vulnerable functions.

Deepbalance was proposed in [42] to address the data imbalance issue where there is an overwhelming amount of non-vulnerable data so that the number of vulnerable instances becomes insufficient. The data imbalance problem poses huge challenges to train effective classifiers for accurate vulnerability detection. The process of building DeepBalance consists of three steps. The first step is to convert source code functions to ASTs in a serialized format, where the ASTs are subsequently tokenized to form AST sequences. The BiLSTM network directly receives AST sequences as input and acts as a feature extractor for automatically obtaining feature representations from input ASTs. In the second step, a novel fuzzy-oversampling method is proposed to mitigate the data imbalance problem by synthetically generating vulnerable code samples based on the learned representations to rebalance the ratio of vulnerable and non-vulnerable samples. In the third step, a random forest algorithm with the feature representations forms a classifier. Experiments are conducted on three open-source projects, including LibTIFF, LibPNG, and FFmpeg. Empirical results confirm the effectiveness of the proposed method.

Game change 1 is constructed based on the availability of sufficient labeled training data collected from the homogeneous sources. However, it is not always possible to guarantee the data source's availability, as real-world vulnerability data is usually scarce. A deep learning-based framework is proposed in [35] to automate software vulnerability discovery. The proposed system in [35] automatically extracts latent vulnerable programming patterns from multiple heteroge-

TABLE 1. Category 1 – Follow-ups of Game Changer 1

Reference	Data Source	Feature/Representation	Neural Network Model	Detection Granularity
Wang et al. [75]	Open source projects	ASTs	DNN (Deep Belief Network)	File-level
Liu et al. [42]	Open source projects	ASTs	BiLSTM	Function-level
Lin et al. [35]	Open source projects and SARD	ASTs & source code	LSTM	Function-level
Nguyen et al. [47]	Open source projects	Statements in source code (opcode and metadata)	Bidrectional RNN	Function-level
Li et al. [29]	Open source projects and SARD	A minimun intermediate representation	Three concatenated CNNs	An interprocedural slices (the subset of code related to vulnerabilities)
Lin et al. [37]	Open source projects	Serialized ASTs	BiLSTM	Function-level
Lin et al. [36]	Open source projects	AST	BiLSTM	Function-level
Russell et al. [59]	SATE TV, Debian, GitHub	Source Code	RNNs and CNNs	Function-level

neous vulnerability-relevant data sources. Two vulnerability-relevant data sources are used to train two network models.

A framework proposed in [35] transfers the knowledge from the existing data source of different forms, aiming to mitigate the problem of lacking vulnerable data. For this purpose, combined with multiple heterogeneous vulnerability-related data sources, a deep learning framework based on the LSTM unit learns the unified vulnerability source code. Its proposed framework consists of four steps. First, the two neural networks are trained on the two data sources, receptively, so that the two trained networks serve as a feature extractor. Second, a set of knowledge representation is obtained by the two feature extractors from the aggregated data. Third, a random forest classifier is trained with the features that combine the two knowledge representations. Fourth, the trained classifier detects the vulnerabilities. This framework deals with two different situations in the process of vulnerability detection. The first situation is a target project without labeled data when the transfer learning method is applied. The second situation is the target software project having a few labeled data samples while detecting vulnerabilities at the function level. Experiments are conducted on the SARD dataset [48] and six open-source projects to evaluate this proposed method's performances. The experiment results are satisfactory. Its performance is better than its counterparts using the generated representations instead of transfer learning. Moreover, the proposed approach outperforms the baseline vulnerability detect systems. Using two networks to learn the representation from two vulnerability-relevant data sources achieves better performance than merely using any single network. Finally, its performance is still comparable when detecting various vulnerability types. However, the proposed method still has some limitations. The first limitation is its incapability of detecting vulnerabilities manifested during the execution process and vulnerabilities across multiple functions or files. The second limitation is that the truncation of various length of input sequences of source code function sequences may cause information loss, which may bias the classifier training.

Code Domain Adaptation Network (CDAN) is proposed in [47] to leverage deep domain adaptation with automatic feature learning for identifying software vulnerability. The CDAN applies transfer learning methods for obtaining knowledge from the labeled projects to process unlabeled projects. It is proposed to address two significant issues. The first issue is to overcome the lack of labeled vulnerability data by adding additional training data to improve the trained model's effectiveness. The second issue is to automatically learn representative features for improving vulnerability detection and prediction capabilities. Furthermore, a semi-supervised variant of CDAN is proposed to fully utilize the unlabeled target data's information by treating the unlabeled target data as the unlabeled component in semi-supervised learning. A fundamental principle in semi-supervised learning is to enforce the clustering assumption. Statements in source code are used as the program representations, where each statement consists of opcode and statement information. Before feeding the data to the bidirectional RNN neural network, there are two data processing steps. The first data processing step standardizes source code by removing comments and non-ASCII characters. The second data processing step embeds the statements into a vector. For evaluating the performance, experiments are conducted on several datasets, including six real-world datasets across six domains. The experiment results indicate that CDAN and its semi-supervised variant outperform VulDeePecker [32], as VullDeePecker does not use domain adaptation.

A method of intelligent vulnerability detection is proposed in [29] utilizing the minimum intermediate representation learning in source code. The detect granularity is an inter-procedure slice that is the subset of code relating to vulnerabilities. It aims to address three problems in the existing intelligent vulnerability detection methods, including the lack of vulnerability data, coarse detection granularity, and long-term dependence due to insufficient vocabulary. The proposed method has four data processing steps. The first step is to exclude useless information and reduce dependency by transferring the data from source code into a minimum intermediate representation. The second step transfers the minimum intermediate representation into a vector so that semantics and structure are retained. The third step obtains high-level features by using the vector as the input to three concatenated CNNs. The last step trains a classifier by using the features learned from the neural networks. Comparative experiments are conducted on the datasets from SARD [48] and open source projects for performance evaluation. The empirical study results show that the proposed method completely outperforms the similarity-based method, the pattern-based method, and a few other baseline methods. However, it suffers from two limitations. The first limitation is its failure to detect vulnerabilities in compiled software because it uses a static analysis paradigm. Its second limitation is its failure to identify or filter out the mislabeled samples in the dataset.

Furthermore, game changer 1 inspires many succeeding studies to apply the idea of representation learning, including “Cross Project Transfer Representation Learning for Vulnerable Function Discovery" [37], “Vulnerability Discovery with Function Representation Learning from Unlabeled Project" [36] and “Automated Vulnerability Detection in Source Code using Deep Representation Learning," [59]. All three papers use representation learning as the core technique.

A method of cross-project transfer representation learning is proposed in [37] for vulnerability discovery. Due to the lack of training data, we have to rely on the features defined by experts. In this case, an individual expert’s opinion may overlook the critical features during the software project’s early stages. The method constructs a BiLSTM neural network to achieve an optimal balance between feature-richness and generalizability. The BiLSTM’s input is the serialized ASTs, and the BiLSTM’s output is the representation indicative of software vulnerability. To train this network, the first step of data processing serializes the semantics of the AST release code while encoding the tokens by continuous Bag-of-Words neural embedding. The method’s final step obtains the neural representation from the existing software projects that are transferred to new projects to enable early vulnerability detection with insufficient training labels. This method’s detection granularity is function-level. The extracted features are deep AST representation. However, the function-level vulnerability ground truth dataset is not publicly available because it uses a self-collected private dataset instead of any public datasets. This method achieves a better precision for predicting vulnerability function than the methods relying on traditional code metrics across multiple projects. However, it cannot be directly applied to the vulnerability that contains multiple functions or multiple files. Furthermore, this method requires further research. The first direction is to explore alternative search strategies to map the AST element as this method uses only a depth-first traversal, e.g., whether the breadth-first traversal achieves a better result. The second direction is to use both the AST-based representation and code metrics as features to train a vulnerability detector.

A method of learning function representation from unlabeled projects is proposed in [36] to facilitate vulnerability discovery. It can detect the vulnerability in the cross-project scenario by leveraging the ASTs learned automatically from the high-level representation of functions. The detect granularity of this method is at the function-level. Besides, a customized BiLSTM neural network is applied to learn the sequential AST representations from raw features. Before feeding training data to the BiLSTM neural network, there are two data processing steps. The first step of data processing is to use the “CodeSensor" parser to extract AST elements from the source code. The second step is to map the AST nodes’ elements to the vector using depth-first traversal technology. For performance evaluation, experiments are conducted on the open-source project data set. The experimental results show that learned representations are beneficial for cross-project vulnerability detection and improve vulnerability detection performance.

A deep representation learning-based approach is proposed in [59] to automate vulnerability detection at the source code level. The vulnerability detection system is developed at the function-level using machine learning by leveraging available codebases in C and C++. Millions of functions are compiled as a supplement dataset in addition to the existing vulnerability dataset with labels. Based on the enhanced dataset, a scalable and fast vulnerability detection tool uses deep feature representation to interpret the compiled source code directly. It no longer needs any complex data processing component as the source code is directly fed to the neural network as input. For performance evaluation, a set of experiments is implemented in the NIST SATE IV benchmark dataset [50] and real software packages. The overall result indicates a promising method for automated software vulnerability detection using deep feature representation learning.

C. FOLLOW-UPS OF GAME CHANGER 2

This subsection reviews two research works on game changer 2, discussing the research problems and the proposed approaches. Table 2 summarizes and compares their key difference in the perspectives of data source, features, neural network models, and detection granularity.

Some limitations of game changer 2 are addressed in [64]. First, game changer 2’s deep learning architecture requires an excessive amount of training data to model security flaws. Second, game changer 2’s method will inevitably overfit synthetic datasets because of the challenges to generate real code massively. Hence, possible solutions are proposed to mitigate these problems using representations of code and deep learning methods to perform arithmetic operations. The code representation can capture appropriate scope information. A code generator named s-bAbI is proposed in [64] to detect buffer overflow vulnerabilities. The generator s-bAbI is implemented by improving a previous work in [4]. In particular, source code is fed into the memory networks to train s-bAbI directly to avoid code analysis. However, the cost is to do some necessary operations beforehand. The first operation is to divide the file into code lines and subsequently tag each line of code. The second operation is to number each line together with a specific mark for identification. The third operation consistently maps tokens to integers. The final operation is to store the integer labels in the array, with zero paddings on the right and bottom. For performance evaluation, many experiments were conducted on the dataset generated by the code generator s-bAbI. The experiment results indicate that the static analysis engine has

TABLE 2. Category 2 – Follow-ups of Game Changer 2

Reference	Data Source	Feature/Representation	Neural Network Model	Detection Granularity
Choi et al. [4]	self-generated	lines of source code	Memory Networks	Function-level
Sestili et al. [64]	code generator: s-bAbI	Source code (consisting of N lines each of which is represented by a list of integer tokens)	Memory Networks	the line-of-code level
Lin et al. [34]	Open source projects	Source code	DNN Text-CNN and four RNN variants: LSTM, GRU, BiLSTM, BiGRU	Dual-granularities: file-level and function-level

high accuracy, but its recall rate is very low. Moreover, the memory networks can achieve similar performance to the static analysis engine. However, it needs enormous training data to achieve high accuracy.

The memory networks model is applied by game changer 2 for detecting buffer overflow vulnerability. However, no systematic performance comparison is provided across different approaches because of its self-constructed/collected dataset. A benchmarking framework is proposed in [34] to assess the effectiveness of deep learning-based vulnerability detectors. In particular, there are six mainstream neural networks, including a DNN network, a text-CNN network, and four RNN variants along with three embedding solutions. A real-world vulnerability ground truth dataset is also built from nine open-source software projects to provide unified performance standard measures since previous vulnerability detection methods are evaluated on self-constructed and self-collected datasets. No code analysis is required because source code is fed to the neural network as direct input for data training. The detection of granularity covers two levels, including function level and file level. There are three modules in the proposed framework. The first module is an embedded module to mark the text/code as an important vector representation. The second module is a training module. The third module is a test module, in which users can use test data as input into the trained network to obtain test results. Experiments are conducted on nine open-source projects. Two baseline systems are using the proposed framework. The first system is built in the self-constructed dataset, which consists of nine open projects to study the neural network's performance in the real-world dataset in the presence of the data imbalance problem. The second baseline system uses the SARD dataset [48] to examine the neural network's performance in the real-world scenario. The experiment results indicate that the systems perform well on the SARD dataset, the DNN network, the Text-CNN, and the BiLSTM network. In the SARD dataset, the neural network model is agnostic for vulnerability detection; but in the real-world dataset, DNN outperforms BiLSTM and Text-CNN. DNN cannot effectively learn potential code vulnerabilities because DNN merely contains a complete contact layer. Conversely, BiLSTM and Text-CNN handle text dependencies well between sequence elements because BiLSTM has a bidirectional LSTM layer, and Text-CNN has many kernels. To further extend this work, vulnerable functions and non-vulnerable functions at the binary-level can be collected and used to train the models. Furthermore, it is crucial for detecting vulnerability to derive the patched functions and files.

D. FOLLOW-UPS OF GAME CHANGER 3

The follow-up studies of game changer 3 are summarized in this subsection, identifying research problems and proposed solutions for vulnerability discovery. Table 3 lists the key difference of the reviewed studies in the perspectives of data source, features, neural network models, and detection granularity.

Three limitations of game changer 3 VulDeePecker [32] are listed in [31]. First, VulDeePecker considers vulnerabilities related to library/API function calls only. Second, VulDeePecker only uses the semantic information contained in the data dependency. Thirdly, VulDeePecker only uses BiLSTM. Hence, six deep neural networks are investigated in [31], including CNN, DBN, LSTM, Gated Recurrent Unit (GRU) [3], BiLSTM, and BiGRU. Nevertheless, it does not strive to explain the reasons for false positives and false negatives. A system named Syntax-based, Semantics-based, and Vector Representations (SySeVR) is proposed in [31] as the first systematic framework to detect vulnerability based on deep learning. SySeVR focuses on solving the problem of obtaining program representations, and it can accommodate semantic and syntax information related to vulnerabilities. The framework uses semantics-based vulnerability candidates (SeVC) as feature representations fed into the network to train. There are three data processing steps: The first step extracts Syntax-based vulnerability candidates (SyVCs) from a training program. The second step transforms SyVCs to SeVCs. The third step transforms SevCs to vectors. The detection granularity is at the SeVC-level, which is multiples lines of code semantically related to each other. Hence, it needs further improvement to precisely detect vulnerabilities. For SySeVR's performance evaluation, experiments are conducted on the open-source projects and the SARD dataset [48]. The experimental results show that SySeVR outperforms some state-of-the-art vulnerability detection methods. The two main limitations of SySeVR are the lack of fine-grained detection and its narrow focus on the detection of vulnerabilities in the C and C++ programming languages.

The VulDeePecker proposed by Li et al. [32] is limited to detect two types of vulnerabilities. A multi-class vul-

TABLE 3. Category 3 – Follow-ups of Game Changer 3

Reference	Data Source	Feature/Representation	Neural Network Model	Detection Granularity
Li et al. [32]	Open source projects and SARD	Code gadgets(only capture data dependence relation)	BiLSTM	slice-level
Li et al. [31]	Open source projects and SARD	SeVC (semantics-based vulnerability candidates)	6 deep neural networks: CNN, DBN, LSTM, GUR, BiLSTM, BiGUR	SeVC (multiples lines of code that semantically related to each other)
Zou et al. [92]	Open source projects and SARD	code gadgets (capturing data dependence relation and control dependence relation)	A novel neural network architecture: building-block LSTM	code gadget (consisting of multiple statements)
Li et al. [30]	Open source projects and SARD	intermediate code	BiRNN-vdl (A novel variant of BiRNN for vulnerability detection and location	program slice-level (leverages program intermediate code to define)
Li et al. [28]	Open source projects	Function names	BiLSTM	Function-level
Liu et al. [43]	Open source projects	program representations (ASTs,Code gadget)	BilSTM	Function-level

TABLE 4. Category 4 – Follow-ups of Game Changer 4

Reference	Data Source	Feature/Representation	Neural Network Model	Detection Granularity
Liu et al. [41]	Open source projects	Binary instructions from each function	Att-BiLSTM (Attention bidirectioal LSTM)	Function-level
Xu et al. [82]	Public source projects	ACFG (attributed control flow graph)	DNN	Function-level
Le et al. [27]	VulDeePecker source code dataset	The opcode and instruction information	RNN	Function-level

nerability detection system named μVulDeePecker [92] is proposed to extend VulDeePecker [32], capable of identifying multiple types of vulnerabilities. μVulDeePecker uses code gadgets as program representations. Code gadgets are a code piece consisting of multiple program statements to provide detailed information on vulnerability types. Code gadgets are used to capture data dependence and control dependence with an LSTM. μVulDeePecker uses four data processing steps. Its first step extracts code gadgets from the training program. Its second step analyzes the standard code gadgets to generate code attentions according to the vulnerability's semantic characteristics. Its third step extracts code attentions. Its fourth step converts the standard code gadgets and code attentions into fixed-length vectors. Finally, these vectors are the input of the LSTM for training. The detection granularity of this paper is a code gadget consisting of many statements. For performance evaluation, experiments are conducted on open-source projects and the SARD dataset [48]. The experimental results show that the method effectively detects multiple types of vulnerabilities. However, this method still has several limitations for further improvement. Its first limitation is its coarse detection granularity as it requires an additional code inspector to precisely locate a vulnerability in a code gadget consisting of many statements. Its second limitation is that μVulDeePecker only detects vulnerability from programs written in C/C++. Its third limitation is μVulDeePecker focuses on detecting vulnerabilities related to library/API function calls. Although VulDeePecker operates on program slices that are more finer-grained than functions, a program slice may have many code lines indicating a low location precision. Coarse-grained vulnerability detection is only a pre-step of vulnerability assessment because it cannot accurately locate vulnerabilities.

The VulDeeLocator is developed by [30] to detect vulnerability by using intermediate code as the program's representation. VulDeeLocator proposes a novel BiRNN-vdl model that stands for the BiRNN for vulnerability detection and location. It requires four data processing steps. First, it extracts the source code- and Syntax-based Vulnerability Candidate (sSyVCs). Second, it locates intermediate code- and Semantics-based Vulnerability Candidate (iSeVCs) based on sSyVCs from intermediate code. Third, it marks the iSeVCs extracted from the training program as vulnerabilities and non-vulnerabilities. Fourth, it converts the iSeVCs along with the labels into representative vectors. For performance evaluation, experiments are conducted on a few open-source projects and the SARD dataset [48] with eleven vulnerability types. The experiment results indicate that VulDeeLocator's method by leveraging intermediate code-based representations is more efficient than using source code-based representation. Besides, BRNN-vdl achieves a high vulnerability location accuracy so that it effectively detects vulnerabilities. However, VulDeeLocator still has a few aspects for further improvement. The first limitation is its narrow scope because VulDeeLocator only detects vulnerabilities in C source code. The second limitation is that VulDeeLocator cannot be used readily and directly because it needs to be compiled into intermediate code.

A lightweight vulnerability discovery method using a deep neural network (LAVDNN) is proposed in [28] to assist vulnerability discovery and provide guidance for manual auditing. LAVDNN aims to analyze unknown software code without restrictions of programming languages, as VulDeePecker [32] can only detect buffer overflow vulnerabilities in C/C++ programs. In particular, LAVDNN uses function names as important semantics features for constructing a deep neural network-based classifier to distinguish functions in source code. LAVDNN has two data processing steps. It obtains semantic features from the open-source program and extracts the function names before vectorizing the function names into a proper representation of the neural network. The experiment results indicate that LAVDNN narrows the scope of analysis and significantly improves code auditing efficiency. However, there are two limitations. The first limitation is that the function names often do not provide enough information, and the second limitation is that LAVDNN requires the complete source code of the program to detect software vulnerabilities at the source code level.

A cross-domain vulnerability discovery method (CD-VulD) is proposed in [43] to mitigate data distribution divergence between training data and testing data. The divergence is caused by training data and testing data that originate from different projects or their different vulnerability types. CD-VulD adopts the concepts of deep learning and deep adaptation (DA) to reduce the divergence between the two distributions so that it detects cross-domain vulnerabilities. Like VulDeepecker, CD-VulD uses code gadgets as the syntax representation of the software program and an Abstract Syntax Tree (AST) as the syntax representation of the software program. The proposed CD-VulD approach contains four data processing steps. Its first step converts software code represented into a token sequence to learn the generalized token embedded in the token. CD-VulD's second step constructs an abstract high-level representation using a deep feature model based on the token sequences. Its third step applies the metric transfer learning framework (MTLF) to learn the cross-domain representation by minimizing the target and source domains' distribution divergence. Its fourth step uses the cross-domain representation to build a vulnerability detection classifier. For performance evaluation, experiments are conducted on a few open-source projects to compare CD-VulD with VulDeePecker and G-VulD. The experimental results show that CD-VulD achieves a better performance in cross-project, cross-vulnerability, and prediction of recent software vulnerabilities than VulDeePecker and G-VulD. Moreover, CD-VulD achieves comparable results in terms of in-domain vulnerability detection. The detection granularity of CD-VulD is at the levels of Abstract Syntax Trees and code gadgets. However, CD-VulD has a few limitations. Its first limitation is that CD-VulD focuses on detecting vulnerabilities in the source code; however, many vulnerabilities appear in the source code of the binary code is not available. Its second limitation is its basic experiment of the C/C++ programming language. Hence, CD-VulD needs to apply another programming language like Java to evaluate performance. Nevertheless, CD-VulD is a CNN-based method instead of an RNN-based method, which needs to be compared in future work performance. Last but not least, it needs to collect more real-world data to achieve better performance comparison.

E. FOLLOW-UPS OF GAME CHANGER 4

There are two pieces of research works which focus on the binary-level vulnerability detection as game changer 4 does. We summarize them in Table 4, showing their difference from the perspectives of data source, features, neural network models, and detection granularity.

A neural network-based cross-platform method is proposed in [82] to detect the similarity of binary code. It aims to improve the existing method's slow detection speed and low detection accuracy while detecting cross-platform binary code similarity. In particular, the more similar between two binary code pieces, the more likely that they originate from the same platform. An attributed control flow graph (ACFG) is used to represent a program fed to a DNN. The proposed system mainly includes two components — ACFG extractor and graph embedded neural network model. Its detection granularity is at the function-level. For performance evaluation, experiments are conducted on public source projects dataset. The experiment results indicate that the proposed method has a better performance than state-of-the-art methods in terms of accuracy of similarity detection, embedding time, and overall training time.

Maximal Divergence Sequential Auto-Encoder (MDSAE) is proposed in [27] for detecting binary software vulnerabilities. MDSAE aims to address the over-reliance on human experts' characteristics manually in the existing binary code vulnerability detection methods. In particular, lacking binaries labeled as either vulnerable or non-vulnerable is the big constrain to binary code vulnerability detection. A binary code vulnerability dataset associated with labels is created along with MDSAE. In MDSAE, opcode and machine instruction information represent a binary code fed into an RNN neural network for data training. MDSAE employs three data processing steps. Its first step detects the entire set of machine instructions by applying the binary disassembly framework. Its second step retrieves the core components containing opcodes and other important information. The third step embeds the instruction information and opcode into a vector. Moreover, the detection granularity of MDSAE is at the function-level. For performance evaluation, experiments are conducted on the VulDeePecker's source code dataset [32]. The experiment results indicate that MDSAE has a better performance than the baseline models.

F. MISCELLANEOUS

Table 5 summarizes and compares two research works that do not belong to any of the categories mentioned above.

A CNN-based approach is proposed in [18] to detect software vulnerability automatically. It aims to widen the scope that existing tools only detect a small and limited subset of possible errors using the manually defined rules. It leverages the available open-source repositories to apply deep learning for discovering vulnerabilities. In particular, CFG represents C/C++ software programs fed to the CNN

TABLE 5. Category 5 – Miscellaneous Papers

Reference	Data Source	Feature/Representation	Neural Network Model	Detection Granularity
Harer et al. [18]	Open source projects	CFG	CNN	Function-level
Niu et al. [49]	Code gadget Database	Taint propagation path	CNN-BiLSTM	File-level

model because CFG works on both source code and binary builds. Its detection granularity is at the function-level. For performance evaluation, experiments are conducted on a few open-source projects. The experiment results indicate that the source-based CNN model has a satisfactory performance. However, the main limitation of this paper is the process of labeling the functions.

A CNN-BiLSTM-based static taint analysis approach is proposed in [49] to identify IoT software vulnerabilities. It aims to address the high false rate problem. This approach locates vulnerabilities automatically by using a combination of deep learning and static analysis algorithms. The taint propagation path is used to represent programs feed into the CNN-BiLSTM neural network for data training. It uses three data processing steps. Its first step sets the rules to select the taint from various files between the source program and their patches. Its second step uses static taint analysis to retrieve the taint propagation paths. Its third step applies the detection model consisting of two-stage BiLSTM to locate and discover software vulnerabilities. Its detection granularity is at the file-level. For performance evaluation, experiments are conducted on the code gadget database. The experiment results indicate that the CNN-based classifier performs better than the baseline models during vulnerability detection.

IV. CHALLENGES AND POSSIBLE FUTURE RESEARCH DIRECTIONS

Based on the review above, the research field of deep learning-based software vulnerability detection is still in its early stage, leaving many problems unsolved. However, it also provides a large open space for future research as large-scale datasets and deep learning models become readily available. Therefore, we draw some conclusive remarks based on the previous works in the perspectives of research challenges and future research directions.

A. INTERPRETABILITY OF DEEP LEARNING MODELS

Deep learning models are naturally challenging to be interpreted because of its layered structure and a massive number of parameters [61]. On the other hand, explainable information of deep learning-based vulnerability detection solutions is critical to stack holders, including end-users, developers, project managers [33]. There are two technical causes for poor interpretability of deep learning methods: nonlinear activation functions and deep neural network structures. Because of the poor interpretability in high-dimensional nonlinear space, most deep neural networks equipped with highly nonlinear activation functions to boost performance suffer from poor interpretability [76]. The black-box nature of the deep neural network models increases the difficulty further [21]. Since software code is complicated, the contemporary vulnerability detection solutions are inherently complex to comprehend the nuance of the code [33].

Many methods have been proposed to explain deep networks to alleviate the interpretability problem of the neural networks [81]. In general, there are three strategies to enhance the model interpretability, including visualization [14], distillation [57], [58], and adding an attention mechanism [10] to the deep neural network model. Some tools are developed and available for public use. For example, LIME [57] is a black-box tool that creates a linear model similar to the original model by using a small amount of data to achieve explainability without sacrificing efficiency; DeepRED [91] extend the CRED [62] algorithm to build a set of decision trees by using RxREN [1] to remove unnecessary inputs and applying the C4.5 [60] algorithm to simplify its decision tree. Last but not least, Razavian et al. [67] propose to understand the ability of CNN layers to solve problems caused by training data.

Deep learning-based solutions should improve the model interpretability, especially for the software vulnerability detection task. The immediate reason is to gain broad acceptance among a wider group of researchers and practitioners as the skepticism over deep learning persists. One first and foremost argument is that deep learning algorithms are vulnerable to backdoors and data poisoning attacks [56]. Although nobody can guarantee that their deep learning model is immune to adversarial attacks [26], [44], [90], deep learning models with enhanced interpretability boost the confidence of users and developers [22]. We must be aware that simple tools like LIME have been attacked by security experts [71], [89]. Hence, we anticipate that future research works will improve the resilience against adversarial attacks while providing interpretable results.

B. APPLICATION SCOPE

In the field of software vulnerability detection based on deep learning, most works surveyed in this paper detect vulnerabilities in source code, such as [31], [32], [35], [37]. These methods do not apply to detect vulnerabilities when source code is unavailable. Furthermore, these methods often do not work well even when a small amount of source code is available because deep learning models require a large amount of data for training [37]. In particular, the scarcity of vulnerable code usually leads to the data imbalance problem [15]. Therefore, we urgently need research to overcome the over-reliance on the availability of source code for vulnerability detection.

With the advancement of AI, new paradigms are proposed to tackle the data imbalance problem. Transfer learning [51] is a novel machine learning paradigm to enable classifiers to work on different datasets. However, only a few works [37] surveyed in this paper apply transfer learning to mitigate the data imbalance problem. One of the biggest challenges for widely applying transfer learning is the high computational expenses required for training deep learning-based vulnerability detectors as hundreds of hours are spent on powerful servers on training some of the current datasets [35], [37]. To leverage the computational resources, federated learning offers a novel approach so that the learning process can be distributed even on some less powerful machines [85].

Future research in deep learning-based vulnerability detection may thrive in manifolds in terms of mitigating data imbalance problems. Although source code with vulnerability is rare, future research may investigate many compiled executable programs in the binary code with known vulnerabilities subject to license agreement and legal conditions [16]. With the help of federated learning, Low-power IoT devices like Raspberry Pi and smartphones could play a significant role in training deep learning models like CNNs or AutoEncoders [8], [25]. Last but not least, detecting security vulnerabilities from emerging blockchain applications is becoming a promising topic [53]. Hence, we anticipate that future research works will analyze binary executable files and apply new paradigms like transfer learning and federated learning.

C. DETECTION GRANULARITY

Many deep learning-based studies reviewed in this survey suffer from a performance bottleneck due to the coarse vulnerability detection granularity. These studies only detect vulnerabilities at function-level, such as [28], [30], [35], [42], [47], some even at file-level, such as [49], [75]. However, the detection granularity at the function- or at file-level may fail to pinpoint the exact location of vulnerable code statement(s) since a vulnerable function or file may have tens of even up to thousands of lines of code. Therefore, fine-grained detection methods can eliminate foreign characteristics in the analysis process while providing detailed auditing information.

Increasing the number of training samples is a strategy to improve the performance of deep learning-based vulnerability detection models. Fortunately, many question and answer (Q&A) sites like Stack Overflow have millions of code snippets. A quantitative approach is proposed in [13] to measure the proliferation of 4,019 security-related code snippets from Stack Overflow. A large number of code clones are detected by [55] from high-quality answers posted on Stack Overflow. Furthermore, the repeatability and reproducibility are investigated in [39] for the deep learning models for detecting software vulnerabilities using the code information retrieved from Stack Overflow. Hence, Stack Overflow offers promising perspectives for improving the detection granularity level of deep learning-based vulnerability detectors.

It is necessary and beneficial for deep learning models to be trained with high-quality data containing vulnerability to improve detection granularity. Real-world software repositories contain billions of code lines with noises, but deep learning models need to be fed with short code snippets. Stack Overflow is the right candidate to retrieve code snippets because there is a steadily increasing number of code snippets posted and verified by users concerning multiple program languages. Hence, we expect future research to build deep learning-based vulnerability detectors with fine granularity levels using Stack Overflow data.

D. LARGE SCALE DATASET WITH GROUND TRUTH

A significant challenge in deep learning-based vulnerability detection is the lack of datasets. The available datasets often cannot be applied in deep learning to train directly because of the need to preprocess data. Although there is a rich source of vulnerability data followed by many vulnerability detection methods, many of them are not publicly available or free for use. Moreover, there is no publicly available vulnerability dataset labeled with the ground truth. Especially for deep learning-based methods, many training data is required to achieve excellent performance. Besides, the data collection process often requires experts to label code laboriously. Hence, the manual software vulnerability collection process is expensive. Some research works attempt to address the challenge of lack of vulnerability data through machine learning techniques [35], [47]. Other works use NLP techniques by applying word embeddings and deep neural networks on language features of vulnerability descriptions to predict vulnerability severity [17]. Nevertheless, lacking datasets with labels has been a challenge for constructing deep learning-based vulnerability detectors.

Self-supervised learning is a new paradigm in machine learning research. One of its main advantages is the ability to generate labels from the training data automatically. This process is called pretraining. And there are multiple pretraining methods for various deep learning architectures, such as DBM [19], CNN [23], RNN and LSTM [5], [86], and many more. Empirical studies in these works show that most labels obtained through the pretraining process are of high quality if we set the training objectives properly. Furthermore, there are readily available tools and solutions to perform pretraining like BERT [10] and ELMo [52]. Despite the extra computational resources of the pretraining process, BERT or ELMo may be a viable solution to generate labels in mass batches.

For generating many labels on software vulnerability data, self-supervised learning may be a practical solution. The research in pretraining language data is much more mature than pretraining software source code or even binary code. Despite the current absence of research outcomes in software vulnerability detection, the success of BERT and ELMo should have given researchers and practitioners confidence and opportunities to explore. We anticipate that pretraining techniques and tools will be applied to label software vulnerability data soon.

E. SEMANTIC INFORMATION PRESERVATION

As pointed out by the surveyed works in [35], [47], semantic information of software code cannot be fully preserved during the stage of data processing. The information loss can be caused by the truncated code sequences in order to form the fixed-length vectors required by the deep learning algorithms. Additionally, to reduce the computational cost and to facilitate the effective learning of contextual information, the code sequences fed to a deep neural network should be kept in a relatively short length. Nevertheless, the truncation of overly long code sequences is inevitable, which also results in the information loss.

Besides, many reviewed studies use ASTs and CFGs derived from source code as features. However, the ASTs and CFGs are not directly used as inputs, instead, they are "flattened" before feeding to neural networks [33]. Namely, the ASTs and CFGs are processed sequentially. This may lose the hierarchical information kept in the tree or graph structure. One of the possible solutions can be utilizing Graph Neural Networks (GNNs) [80] to process the ASTs, CFGs, and other graph-based program representations. Hence, applying graph neural networks to process structural program representations for software vulnerability detection may be an interesting future research direction.

V. CONCLUSION

This paper reviews recent studies applying deep learning for vulnerability detection and identifies four cornerstones in this field. We call these four cornerstones as four game changers because they provide novel ideas and approaches by bringing fresh air to the field of deep learning-based vulnerability detection. According to four game changers, we categorize relevant research works to provide an organized review for researchers in this emerging field. This survey provides an understanding of vulnerability detection achievements and research trends based on deep learning and future research directions. Our main conclusion is that the application of deep learning techniques for software vulnerability analysis and discovery is not yet mature through the review of existing studies. With the rapid development of data-driven techniques, significant advances in machine learning and deep learning will continuously increase vulnerability detection value. This developing but rapidly growing field will inspire and attract more researchers to contribute to it.

ACKNOWLEDGMENT

This research was supported by the Optoelectronic Information Technology Key Laboratory Open Project Fund of Yunnan Province, China (YNOE-2020-01) and the 2019 Educational Research Project of Fujian Province, China (KC20008P).

REFERENCES

[1] M. G. Augasta and T. Kathirvalavakumar, "Reverse engineering the neural networks for rule extraction in classification problems," Neural Processing Letters, vol. 35, no. 2, pp. 131–150, 2012.
[2] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android hiv: A study of repackaging malware for evading machine-learning detection," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 987–1001, 2020.
[3] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," arXiv preprint arXiv:1406.1078, 2014.
[4] M.-j. Choi, S. Jeong, H. Oh, and J. Choo, "End-to-end prediction of buffer overruns from raw source code via neural memory networks," arXiv preprint arXiv:1703.02458, 2017.
[5] A. Conneau and G. Lample, "Cross-lingual language model pretraining," in Advances in Neural Information Processing Systems, 2019, pp. 7059–7069.
[6] R. Coulter, Q. Han, L. Pan, J. Zhang, and Y. Xiang, "Data-driven cyber security in perspective–intelligent traffic analysis." IEEE Transactions on Systems, Man, and Cybernetics, pp. 1–13, 2019.
[7] ——, "Code analysis for intelligent cyber systems: A data-driven approach," Information Sciences, vol. 524, pp. 46–58, 2020.
[8] R. Coulter and L. Pan, "Intelligent agents defending for an iot world: A review," Computers & Security, vol. 73, pp. 439–458, 2018.
[9] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction," arXiv preprint arXiv:1708.02368, 2017.
[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019, pp. 4171–4186.
[11] F. Dong, J. Wang, Q. Li, G. Xu, and S. Zhang, "Defect prediction in android binary executables using deep neural network," Wireless Personal Communications, vol. 102, no. 3, pp. 2261–2285, 2018.
[12] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," Scientific Programming, vol. 2019, pp. 1–14, 2019.
[13] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P). IEEE, 2017, pp. 121–136.
[14] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 3429–3437.
[15] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," ACM Computing Surveys, vol. 50, no. 4, p. 56, 2017.
[16] D. Gibert, C. Mateu, and J. Planes, "Hydra: A multimodal deep learning framework for malware classification," Computers & Security, p. 101873, 2020.
[17] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017, pp. 125–136.
[18] J. A. Harer, L. Y. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, L. H. Hamilton, G. I. Centeno, J. R. Key, P. M. Ellingwood et al., "Automated software vulnerability detection with machine learning," arXiv preprint arXiv:1803.04497, 2018.
[19] G. E. Hinton and R. R. Salakhutdinov, "A better way to pretrain deep boltzmann machines," in Advances in Neural Information Processing Systems, 2012, pp. 2447–2455.
[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[21] A. Holzinger, "From machine learning to explainable ai," in Proceedings of the 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA). IEEE, 2018, pp. 55–66.
[22] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," Computer Science Review, vol. 37, p. 100270, 2020.
[23] Y. Huang, R. Wu, Y. Sun, W. Wang, and X. Ding, "Vehicle logo recognition system based on convolutional neural networks with a pretraining strategy," IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 4, pp. 1951–1960, 2015.
[24] T. Ji, Y. Wu, C. Wang, X. Zhang, and Z. Wang, "The coming era of alphahacking?: A survey of automatic software vulnerability detection,

exploitation and patching techniques," in Proceedings of the 2018 IEEE 3rd International Conference on Data Science in Cyberspace (DSC). IEEE, 2018, pp. 53–60.

[25] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," Digital Investigation, vol. 24, pp. S48–S59, 2018.

[26] R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia, "Adversarial machine learning–industry perspectives," arXiv preprint arXiv:2002.05646, 2020.

[27] T. Le, T. Nguyen, T. Le, D. Phung, P. Montague, O. De Vel, and L. Qu, "Maximal divergence sequential autoencoder for binary software vulnerability detection," in Proceedings of the 2018 International Conference on Learning Representations, 2018.

[28] R. Li, C. Feng, X. Zhang, and C. Tang, "A lightweight assisted vulnerability discovery method using deep neural networks," IEEE Access, vol. 7, pp. 80 079–80 092, 2019.

[29] X. Li, L. Wang, Y. Xin, Y. Yang, and Y. Chen, "Automated vulnerability detection in source code using minimum intermediate representation learning," Applied Sciences, vol. 10, no. 5, p. 1692, 2020.

[30] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu, and H. Jin, "Vuldeelocator: A deep learning-based fine-grained vulnerability detector," arXiv: Cryptography and Security, 2020.

[31] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, S. Wang, and J. Wang, "Sysevr: A framework for using deep learning to detect software vulnerabilities," arXiv preprint arXiv:1807.06756, 2018.

[32] 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," in Proceedings of the 2018 NDSS, 2018.

[33] G. Lin, Q. H. Sheng Wen, and Y. X. Jun Zhang, "Software vulnerability detection using deep neural networks: A survey," Proceedings of the IEEE, pp. 1–24, 2020, DOI: 10.1109/JPROC.2020.2993293.

[34] G. Lin, W. Xiao, J. Zhang, and Y. Xiang, "Deep learning-based vulnerable function detection: A benchmark," in Proceedings of the 2019 International Conference on Information and Communications Security. Springer, 2019, pp. 219–232.

[35] G. Lin, J. Zhang, W. Luo, L. Pan, O. De Vel, P. Montague, and Y. Xiang, "Software vulnerability discovery via learning multi-domain knowledge bases," IEEE Transactions on Dependable and Secure Computing, 2019, DOI: 10.1109/TDSC.2019.2954088.

[36] G. Lin, J. Zhang, W. Luo, L. Pan, and Y. Xiang, "Poster: Vulnerability discovery with function representation learning from unlabeled projects," in Proceedings of the 2017 ACM SIGSAC Conference on Computer & Communications Security (CCS). ACM, 2017, pp. 2539–2541.

[37] G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, O. De Vel, and P. Montague, "Cross-project transfer representation learning for vulnerable function discovery," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3289–3297, 2018.

[38] B. Liu, L. Shi, Z. Cai, and M. Li, "Software vulnerability discovery techniques: A survey," in Proceedings of the 2012 4th International Conference on Multimedia Information Networking and Security. IEEE, 2012, pp. 152–156.

[39] C. Liu, C. Gao, X. Xia, D. Lo, J. Grundy, and X. Yang, "On the replicability and reproducibility of deep learning in software engineering," arXiv preprint arXiv:2006.14244, 2020.

[40] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," IEEE Communications Surveys & Tutorials, vol. 20, no. 2, pp. 1397–1417, 2018.

[41] S. Liu, M. Dibaei, Y. Tai, C. Chen, J. Zhang, and Y. Xiang, "Cyber vulnerability intelligence for internet of things binary," IEEE Transactions on Industrial Informatics, vol. 16, no. 3, pp. 2154–2163, 2020.

[42] S. Liu, G. Lin, Q. Han, S. Wen, J. Zhang, and Y. Xiang, "Deepbalance: Deep-learning and fuzzy oversampling for vulnerability detection," IEEE Transactions on Fuzzy Systems, vol. 28, no. 7, pp. 1329–1343, 2020.

[43] S. Liu, G. Lin, L. Qu, J. Zhang, O. D. Vel, P. Montague, and Y. Xiang, "Cd-vuld: Cross-domain vulnerability discovery based on deep domain adaptation," IEEE Transactions on Dependable and Secure Computing, 2020, DOI: 10.1109/TDSC.2020.2984505.

[44] X. Ma, Y. Niu, L. Gu, Y. Wang, Y. Zhao, J. Bailey, and F. Lu, "Understanding adversarial attacks on deep learning based medical image analysis systems," Pattern Recognition, p. 107332, 2020.

[45] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, vol. 27, pp. 504–518, 2015.

[46] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in Proceedings of the 14th ACM SIGSAC Conference on Computer & Communications Security (CCS). ACM, 2007, pp. 529–540.

[47] V. Nguyen, T. Le, T. Le, K. Nguyen, O. DeVel, P. Montague, L. Qu, and D. Phung, "Deep domain adaptation for vulnerable code function identification," in Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019, pp. 1–8.

[48] NIST, "Software assurance reference dataset project," https://samate.nist.gov/SRD/, 2018, accessed: 2020-07-20.

[49] W. Niu, X. Zhang, X. Du, L. Zhao, R. Cao, and M. Guizani, "A deep learning based static taint analysis approach for iot software vulnerability location," Measurement, vol. 152, p. 107139, 2020.

[50] V. Okun, A. Delaitre, and P. E. Black, "Report on the static analysis tool exposition (sate) iv," NIST Special Publication, vol. 500, p. 297, 2013.

[51] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, 2009.

[52] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2018, pp. 2227–2237.

[53] P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss, "Security analysis methods on ethereum smart contract vulnerabilities: A survey," arXiv preprint arXiv:1908.08605, 2019.

[54] D. RadjenoviÄĞ, M. HeriÄ■ko, R. Torkar, and A. Å¡ivkoviÄ■, "Software fault prediction metrics: A systematic literature review," Information and Software Technology, vol. 55, no. 8, pp. 1397–1418, 2013.

[55] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic code snippets on stack overflow," IEEE Transactions on Software Engineering, pp. 1–1, 2019, DOI: 10.1109/TSE.2019.2900307.

[56] K. Ren, T. Zheng, Z. Qin, and X. Liu, "Adversarial attacks and defenses in deep learning," Engineering, vol. 6, pp. 346–360, 2020.

[57] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.

[58] ——, "Anchors: High-precision model-agnostic explanations," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018.

[59] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated vulnerability detection in source code using deep representation learning," in Proceedings of the 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2018, pp. 757–762.

[60] S. L. Salzberg, "Book review: C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993," Machine Learning, vol. 16, no. 3, pp. 235–240, 1994.

[61] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," arXiv preprint arXiv:1708.08296, 2017.

[62] M. Sato and H. Tsukimoto, "Rule extraction from neural networks via decision tree induction," in Proceedings of the International Joint Conference on Neural Networks (IJCNN'01), vol. 3. IEEE, 2001, pp. 1870–1875.

[63] "Record-breaking number of vulnerabilities disclosed in 2017: Report," https://www.securityweek.com/record-breaking-number-vulnerabilities-disclosed-2017-report, SecurityWeek, February, 2018, accessed: 2018-05-28.

[64] C. D. Sestili, W. S. Snavely, and N. M. VanHoudnos, "Towards security defect prediction with ai," arXiv preprint arXiv:1808.09897, 2018.

[65] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," IEEE Transactions on Dependable and Secure Computing, vol. 12, no. 6, pp. 688–707, 2015.

[66] L. K. Shar and H. B. K. Tan, "Predicting common web application vulnerabilities from input validation and sanitization code patterns," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 310–313.

[67] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 806–813.

[68] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of soft-

ware vulnerabilities," IEEE Transactions on Software Engineering, vol. 37, no. 6, pp. 772–787, 2011.

[69] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" Empirical Software Engineering, vol. 18, no. 1, pp. 25–59, 2013.

[70] S. K. Singh and A. Chaturvedi, "Applying deep learning for discovery and analysis of software vulnerabilities: A brief survey," in Soft Computing: Theories and Applications. Springer, 2020, pp. 649–658.

[71] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, "Fooling lime and shap: Adversarial attacks on post hoc explanation methods," in Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, 2020, pp. 180–186.

[72] S. Sukhbaatar, J. Weston, R. Fergus et al., "End-to-end memory networks," in Advances in Neural Information Processing Systems, 2015, pp. 2440–2448.

[73] N. Sun, J. Zhang, P. Rimba, S. Gao, L. Y. Zhang, and Y. Xiang, "Data-driven cybersecurity incident prediction: A survey," IEEE Communications Surveys and Tutorials, vol. 21, no. 2, pp. 1744–1772, 2019.

[74] D. Votipka, R. Stevens, E. Redmiles, J. Hu, and M. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," in Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P). IEEE, 2018, pp. 374–391.

[75] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in Proceedings of the 38th International Conference on Software Engineering (ICSE). ACM, 2016, pp. 297–308.

[76] X. Wang, X. He, F. Feng, L. Nie, and T.-S. Chua, "Tem: Tree-enhanced embedding model for explainable recommendation," in Proceedings of the 2018 World Wide Web Conference, 2018, pp. 1543–1552.

[77] Y. Wang, P. Jia, L. Liu, and J. Liu, "A systematic review of fuzzing based on machine learning techniques," arXiv: Cryptography and Security, 2019.

[78] J. Weston, S. Chopra, and A. Bordes, "Memory networks," arXiv preprint arXiv:1410.3916, 2014.

[79] ——, "Memory networks," arXiv preprint arXiv:1410.3916, 2014.

[80] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," IEEE Transactions on Neural Networks and Learning Systems, pp. 1–21, 2020, DOI: 10.1109/TNNLS.2020.2978386.

[81] N. Xie, G. Ras, M. van Gerven, and D. Doran, "Explainable deep learning: A field guide for the uninitiated," arXiv preprint arXiv:2004.14545, 2020.

[82] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in Proceedings of the 2017 ACM SIGSAC Conference on Computer & Communications Security (CCS, 2017.

[83] F. Yamaguchi, F. Lindner, and K. Rieck, "Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning," in Proceedings of the 5th USENIX Conference on Offensive Technologies. USENIX Association, 2011.

[84] F. Yamaguchi, M. Lottmann, and K. Rieck, "Generalized vulnerability extrapolation using abstract syntax trees," in Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC). ACM, 2012, pp. 359–368.

[85] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," ACM Transactions on Intelligent Systems and Technology (TIST), vol. 10, no. 2, pp. 1–19, 2019.

[86] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in Advances in Neural Information Processing Systems, 2019, pp. 5753–5763.

[87] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," IEEE ACM Transactions on Networking, vol. 23, no. 4, pp. 1257–1270, 2015.

[88] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," IEEE Transactions on Parallel and Distributed systems, vol. 24, no. 1, pp. 104–117, 2013.

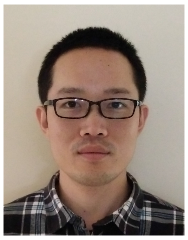
[89] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang, "Interpretable deep learning under fire," in Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), 2020.

[90] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," arXiv preprint arXiv:2004.11573, 2020.

[91] J. R. Zilke, E. L. Mencía, and F. Janssen, "Deepred–rule extraction from deep neural networks," in Proceedings of the International Conference on Discovery Science. Springer, 2016, pp. 457–473.

[92] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "μvuldeepecker: A deep learning-based system for multiclass vulnerability detection," IEEE Transactions on Dependable and Secure Computing, pp. 1–1, 2019, DOI: 10.1109/TDSC.2019.2942930.

PENG ZENG is working towards his master's degree in the School of Physics & Electronic Information in the Yunnan Normal University, China. His research interests include the use of deep learning and vulnerability data analysis for software vulnerability detection.



GUANJUN LIN received the Ph.D degree in the School of Software and Electrical Engineering at the Swinburne university of technology, Melbourne, VIC., Australia, in 2019. He is current a lecturer at School of Information Engineering, Sanming University. His research interest is the application of deep learning techniques for software vulnerability detection.

LEI PAN (M'12) received the Ph.D. degree in computer forensics from Deakin University, Australia, in 2008. He is currently a Senior Lecturer with the School of Information Technology, Deakin University. His research interests are cyber security and privacy. He has authored 50 research papers in refereed international journals and conferences, such as the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.

YONGHANG TAI received his Ph.D. degree in computer science in 2019 of IISRI from Deakin University, Geelong, VIC, Australia. He is now an associate professor in the School of Physics & Electronic Information in the Yunnan Normal University, China. His current research interests include physics-based simulation, applied AI, medical AR/MR and precision medicine.

JUN ZHANG is a professor in the School of Physics & Electronic Information in the Yunnan Normal University, China. His research interests include applied AI, big data analytics, precision medicine and healthcare digital twin. He has published over 100 research papers in refereed international journals and conferences. He has served as chairs in many international conferences.