International Journal of Scientific Research and Management (IJSRM)
||Volume||13||Issue||06||Pages||2254-2275||2025||
Website: https://ijsrm.net ISSN (e): 2321-3418
DOI: 10.18535/ijsrm/v13i06.ec05

Network Embedding Techniques for Predicting Software Defects: A Review

Sweta Mehta[1], Pankaj K. Goswami[1], K.Sridhar Patnaik[2]

[1] Department of CSE, Sarala Birla University, Ranchi, India
[2] Department of CSE, Birla Institute of Technology, Mesra, Ranchi, India

Abstract
In the software development process, ensuring the quality of the software is essential. Software defect prediction (SDP) is of significant importance in identifying software modules with a high likelihood of defects. Several machine learning-based defect prediction models have been developed and implemented in recent years. Researchers have also utilized network embedding for SDP, showcasing the adaptability of Natural Language Processing techniques within the domain of defect prediction. This study aims to review, investigate, and discuss network embedding's use in SDP. We examined the previous 15 years' defect prediction articles using network embedding, the majority of which were published in notable conferences and software engineering journals. Each network embedding technique, its findings, and its particular roles in SDP have been described in detail. The papers that have been reviewed are listed in the order of publication along with their comparative assessment. We have developed three research questions that emphasize the significance of analyzing network representations, particularly network embedding, for identifying potential software defects. According to our knowledge, this review is the first to include a thorough analysis of both the transductive and inductive variants of network embedding, along with their potential in machine learning (ML) for predicting software defects. This article extensively explores the challenges and puts forth potential research directions as solutions, intending to effectively guide future research efforts for academics and practitioners in the field of SDP.

Keywords: Software Defect Prediction, Network Embedding, Machine Learning, Software Dependency Network, Graph Neural Network, Network Analysis

1 Introduction

Software testing is a pivotal aspect of the software development life cycle as it is an essential safety measure before the release of software in the market. Before any software is released after development or maintenance fixes, it must be tested for possible defects. To examine every source code file, however, would be impractical due to both time and resource constraints. By employing software defect prediction models, it becomes possible to predict the modules that are more likely to have defects, thus helping the software testing teams to focus on modules having a substantial tendency for defects and giving testers valuable guidance (Alharthi et al., 2021; Hossain and Chen, 2022). The recent SDP models mainly constitute the defect data set and machine learning algorithms. The defect data set comprises software defect metrics generated from previous projects stored in repositories and fed into machine learning binary classifier algorithms that predict defect-prone software modules. Although numerous techniques have been created to predict defects, it has proven a challenge to consistently produce reliable results for increasingly complex software being developed currently. Determining a reliable, accurate, cost- and time-effective method to identify defect-prone modules is thus still an ongoing challenge in software engineering.

Static code metrics provide estimations of software characteristics related to defect propensity and, consequently, to the level of quality. Size is one such feature that is frequently measured using LOC counts, and readability is measured using operand and operator counts. This category of static metrics is defined as Halstead metrics (Halstead, 1977). Metrics for measuring the complexity of code are provided by McCabe feature metrics (McCabe, 1976). With the development of object-oriented software, CK metrics were

developed (Jureczko and Spinellis, 2010). MOOD feature metrics (Harrison et al., 1998) have also proved to be useful in capturing object-oriented features of the software.

Networks serve as a common representational form for complex systems, encompassing social interactions, relationships among various biological components, and informational networks. It is widely acknowledged that network information is inherently complex and challenging to manage. Overcoming the first critical hurdle in network information processing involves finding an efficient representation method that enables proficient execution of advanced analytical tasks such as pattern discovery, analysis, and prediction. Taking advantage of class dependency networks for SDP signifies an important development in software engineering field. Dependency networks correspond to software modules as nodes, and interactions between each module are expressed as edges. These Software networks are then analyzed to obtain network metrics, which use the interdependence relationship between individual code modules to define the network's structural properties. Recent studies have shown that network metrics are capable of predicting software defects (Zimmermann & Nagappan, 2008; Tosun et al., 2009; Nguyen et al., 2010; Premraj and Herzig, 2011; Ma et al., 2016; Chen et al., 2016). Network metrics consist of ego and global network metrics, both of which can be used to assess the nodes (modules) of a software network (Chen et al., 2016, Gong et al., 2021). To create a defect prediction system with enhanced accuracy, Gong et al. (2021) integrated network metrics obtained through social network analysis from the software's dependency network with static code metrics. However, network metrics that the aforementioned researchers have used for SDP still fall under the category of traditional features. With new developments in deep learning technologies, researchers have shown interest in developing network representation methods that automatically learn the features and hidden characteristics of network nodes (Hamilton et al., 2017). Network Embedding is one of the methods to represent a network. As a promising method of network representation, network embedding could assist in applications that require network visualization, node classification, link prediction, and node clustering (Goyal & Ferrara, 2018). Diverse computational methods like random walk (traversing a graph by taking steps to neighboring nodes based on a stochastic process), matrix factorization (decomposing the adjacency matrix of a network into matrices of lower dimension), and deep learning are used to generate the representations. These methods generally seek to maximize an objective function to identify or retain essential network features. In Network Embedding, the nodes in the network are converted to vector representations in a lower dimensional space in a manner that preserves their respective structural information. Unlike network embedding approaches, which instantly comprehend the structural characteristics of software networks, network metrics are limited to analyzing software networks statically. Therefore, obtaining structure-related data from code modules using network embedding techniques and subsequently utilizing the learned characteristics for SDP may boost the accuracy of current prediction models.

1.1 Contributions to the Research

Various models for early defect prediction associated with within-project as well as cross-project defect predictions have been studied and presented by researchers as the significance of identifying defects in software modules has increased exponentially. To create SDP models in the initial research, conventional software metrics were used. These models were dependent on metrics, which include Halstead metrics (Halstead, 1977), McCabe metrics (McCabe, 1976), MOOD metrics (Harrison et al., 1998), CK metrics (Jureczko and Spinellis, 2010), etc. In addition, the industry has been dominated for the past 15 years by the object-oriented way of to software development. Therefore, the researchers transformed their technique for developing the models to object-oriented metrics. With the development of CK metrics, structural characteristics, complexity, and size of modules of object-oriented software were used as features of the software for SDP. The performance of these metrics outperformed conventional metrics in defect prediction (Coscia et al., 2012). But, these metrics are unable to accurately reflect the complexity of modern software. While MOOD metrics offer a global assessment of the software system, CK metrics seem to deal with software evaluation at the class level. The dependability of object oriented metrics is also impacted by the size of the software.Further, Network metrics have also been utilized by some researchers to assess defect prediction performance. However, the number of studies incorporating network metrics is comparatively lower than those focusing on object-oriented metrics. This is primarily due to the prevalence of defect datasets that predominantly include object-oriented metrics. Consequently, our work aims to evaluate the utilization of network embedding in comparison to research that relies on object-oriented metrics. However,

the use of object-oriented parameters poses a challenge in terms of the predictability of the output generated by the prediction model. This unpredictability arises from the use of different tools for collecting the metrics data. Chidamber and Kemerer Java Metrics, Understand, Eclipse Metrics Plugin, SourceMeter, Jarchitect, Radon, etc., are some of the popularly used tools for generating software metrics. Each tool may generate slightly different metrics data for the same software, leading to unreliable accuracy in defect prediction. This variation in software metrics data across various tools is due to the different algorithms used in the tool implementation. These tools support different programming languages to varying extents or may interpret code constructs in different ways. The use of software metrics also faces difficulty in the following: 1) capturing all relevant information using software metrics becomes challenging due to the intricate relationships between multiple interacting components of the software; 2) the impact of a class, dependency, or relationship on the entire system; 3) correlations between classes that play different kinds of roles as a system expands and transforms during development. Recognizing these vulnerabilities, we were motivated to develop an SDP model that is independent of the software metrics obtained from various tools.

We were motivated to write this review article due to the increasing significance of network embedding in various software engineering applications (Zeng et al., 2021; Bahaweres et al., 2021; Dong et al., 2019). We recognize that network embedding has the potential to create SDP models that are not reliant on software metrics. The demand for such models to exhibit improved performance is critical, as they are essential for accurately predicting defects in software engineering practices. Contrary to previous research, our work introduces network embedding techniques for automatic feature vector learning in defect prediction. We aim to provide empirical evidence demonstrating that these techniques can achieve satisfactory performance compared to existing models. In summary, our contributions to the research field are as follows:

- This study reviews the network embedding research papers from the last 15 years. Well-sampled research papers were used to create this article.

- We have thoroughly conducted a study to discuss the various network embedding methods used for SDP. Additionally, we have categorized these methods into inductive and transductive approaches.

- In addition, we have compiled a comparative analysis of network embedding methods used for SDP and compared their benefits and drawbacks.

- The study addresses the existing challenges and outlines the future research directions that can improve the existing SDP models.

We have framed the following three research questions to outline our work.
(a) RQ1: How can the accuracy of defect prediction be increased by using different network representations of software projects?
(b) RQ2: Which is the best methodology to be employed for network analysis in the context of SDP?
(c) RQ3: What are the best techniques utilized for network embedding for predicting software defects?

1.2 Organization of the Survey

This paper has been divided into the following sections. We define the fundamentals and related works on SDP, software network analysis, and network embedding in Section 2 followed by Section 3 which describes the major categories of network embedding. We identify the categories of existing research based on embedding procedures. Observations concerning each technique are retrieved, and a comprehensive review of different embedding methods is put together. Section 4 focuses on the research methodology used for this review article. Further, we provide the applications of network embedding in software defect prediction in the 5th Section. In Section 6, we talk about our study's findings. Validity threats are mentioned in Section 7. A summary of this work and future directions is discussed in Section 8.

2 Background and related work

In this segment, background information is presented. It includes relevant research conducted on SDP, software metrics, class dependency networks, network analysis, and network embedding approaches.

2.1 Software Defect Prediction

Quality control for software has extensively utilized SDP techniques, which can greatly reduce software development costs. SDP involves creating a predictive model using historical defect data, which determines the likelihood of a software module being defect-prone. The typical process of software defect prediction involves two stages. During the initial stage, defect data and code metrics are gathered from the software source code. The second stage involves classifying the modules using machine learning algorithms. These algorithms are designed to create an accurate model that generates consistent results with enhanced prediction performance. The objective is to develop a reliable and efficient approach for identifying defect-prone modules. By incorporating SDP techniques into quality control processes, organizations can proactively identify potential software defects and take preventive measures, leading to improved software reliability and cost savings in the development lifecycle. The research work on SDP can be broadly categorized into the following categories, based on the scope and context of prediction.

1. Within-project SDP: This approach utilizes information from the same project to construct the defect prediction model. The datasets used for training and testing purposes are generated from the current project (A.b) for which the prediction model is being developed. Here, 'A' refers to the project name, and 'b' indicates the version of the project. The model will be built based on the defect dataset obtained from 'A.b'.
2. Cross-version SDP: In this approach, data from earlier versions of the same project is used. The training dataset is constructed using information from previous versions, while the test dataset includes data from the current version. For instance, the prediction model for project 'A.b' is trained using data from the previous version 'A.a,' and then the trained model is tested on 'A.b'.
3. Cross-project SDP: Cross-project SDP involves building the prediction model using data from a project and then using the model to predict defects on another project. For example, to develop a prediction model for project ‘B’, the training dataset is constructed using data from project ‘A’, while the test dataset is generated using data from project ‘B’. This approach explores the transferability of defect prediction models across different projects.

Numerous previous studies have made substantial use of static features (Premraj and Herzig, 2011). These studies involve examining the code of software modules to derive statistical features and extract traditional metrics. Process metrics are generated by evaluating code changes made in each version of the software module, providing insights into the software development process. Software metrics have frequently been used to predict defects for software projects (Gong et al., 2021). Since the early nineties, the object-oriented (OO) methodology has become very common in the software development industry. Numerous metrics, including the MOOD metrics set and the CK metrics, have been proposed by researchers to guarantee the quality of OO software. These metrics can be used in a variety of ways for real-world projects, and they have been validated by successful software development projects. Defect prediction based on network metrics (Ma et al., 2016; Fan et al., 2019), leverages class dependency networks for obtaining the metrics. In a study by Qu et al. (2021), class dependency networks and k-core decomposition (k = degree of node) were used to highlight that classes with larger values of k exhibit a higher probability of defects. For SDP, a variety of machine learning techniques have been studied, such as Support Vector Machine, Neural Network, Naive Bayes, Bayesian Belief Network, Decision Trees, Random Forest as well as ensemble learning (Alharthi et al., 2021; Gurung, 2022). The common strategy used in current techniques is to extract latent structural and semantic characteristics from the software source code (Bahaweres et al., 2021; Huo et al., 2018; Yang et al., 2023).

2.2 Software Network Analysis

A graph or network data type is a significant type mainly used in our daily lives and for academic activity. Research activities have recently focused on developing network applications using machine learning (Hou et al., 2020). Some include anomaly detection, link prediction (Li et al., 2018; Du et al., 2022), node embedding, etc. All these applications make use of network analysis (Wang et al., 2022; Pinzger et al., 2008; Yang et al., 2018). Networks offer the most suitable framework for analyzing the structure and components of complex systems, such as software projects. Class dependency networks, software architecture maps, function call graphs, and software class diagrams are some of the networks that can be used to represent software systems.The conventional method of network representation presents several challenges when it comes to evaluating and interpreting large networks.One of the major issues faced while network analysis is determining how to mathematically represent a network as the space of interactions between nodes within a

network is enormous because of the heterogeneous properties of edges and nodes. It is challenging to separate graph or network data into smaller samples than other data types. The adjacency matrix, which is a square matrix with a size the same as the node count of the network, is a usual and basic representation technique of a network (Hamilton et al., 2017).

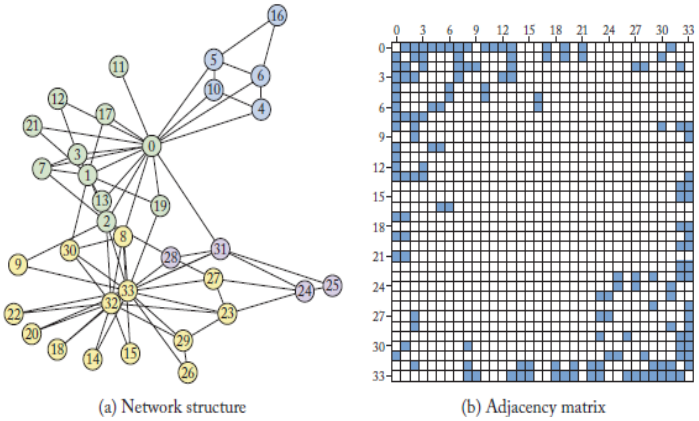


Fig. 1 A sample network structure and its adjacency matrix. Zero entries are indicated using white blocks (Yang et al., 2021)

The major disadvantage of using an adjacency matrix for graph data representation is high dimensionality and sparse representation of data. High dimensionality increases the computation cost. Some of the conventional approaches also include multidimensional scaling, IsoMap, Laplacian Eigenmap, etc. which typically require affinity graph construction and solving eigenvectors. A major drawback of these techniques is that they are not scalable. In addition, these methods lead to high computational burdens and make it challenging to design and implement parallel and decentralized algorithms. It is also difficult to use advanced machine learning algorithms with conventional network data as these methods generally require input data to be presented as individual, independent vectors. Due to these drawbacks, various alternative network representations have been proposed, and network embedding is one such technique that has the capability to address a variety of network analysis and processing tasks (Cai et al., 2018; Qu et al., 2021).

2.3 Network Embedding

Network Embedding (NE) technique encodes every node/vertex belonging to the network/graph into a real-valued low-dimensional vector form (Xie et al., 2021). This is done to represent the network nodes in a manner such that it retains the basic properties of the network i.e. the geometric representation of the embedded vectors reflects the original relationships among the nodes of the network. This representation not only reduces the complexity involved in representing large graphs/network data but also has the additional benefit of automatically learning the features of graph/network. The encoding for the network is also effective in distinguishing the interaction between the nodes having various attributes. This can be defined as, for a network $N = (V, E)$, where V denotes vertices while E denotes edges, a network embedding can be represented as a mapping f: $v_i \rightarrow y_i \in \mathrm{R}^d$, $\forall\ v_i \in V$ given that $d << |V|$ and the proximity measure for network represented by N is preserved by f, the mapping function (Goyal & Ferrara, 2018). Proximity measure refers to a function that indicates the similarity or the distance between a pair of nodes (Cai et al., 2018).

First-order proximity: It is the direct link between two network nodes. It is defined as the local proximity between the nodes/vertices and is thus efficient in representing the local structural information of the network/graph.

Second-order proximity: It defines the extent of the resemblance of the neighboring structures between two nodes/vertices of a network/graph.

The overview of network embedding framework is depicted in Fig. 2. Some popular techniques proposed for NE include node2vec (Grover and Leskovec, 2016), LINE (Tang et al., 2015), DeepWalk (Perozzi et al., 2014), etc. These are categorized under the distributional hypothesis, which focuses on the fact that nodes with high context similarity indexes are similar. To learn the relevant embeddings all the characteristics of the entities i.e. the nodes and edges need to be captured. Fig. 3 presents an illustration of network embedding using DeepWalk algorithm. As indicated in Fig. 3, DeepWalk takes input in the form of a graph

(Fig. 3a) and generates representations of the graph's vertices with two latent dimensions (Fig. 3b). The categorization of the network embedding techniques has been discussed in the following section.

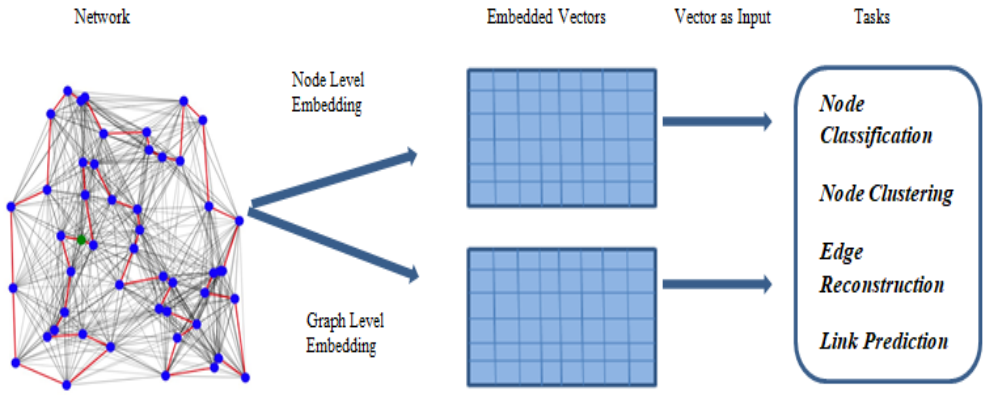
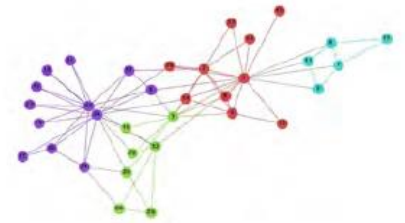
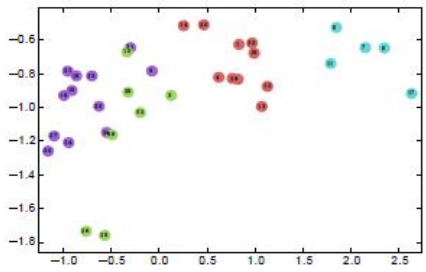


Fig. 2 A basic network embedding framework



(a) Input: karate network



(b) Output: representations

Fig. 3 A karate network used for an illustration of network embedding. The snapshots were taken from the DeepWalk algorithm (Perozzi et al., 2014)

3 Network Embedding Techniques

This section contains the different approaches for embedding networks on the basis of the methodologies employed. The network embedding technique is for representing network data in a low-dimensional vector space while retaining as much network data as feasible (Li et al., 2022; Chen et al., 2018; Pereira et al., 2019; Dai et al., 2019). The network embedding techniques differ mainly in how they understand the relationship between the network nodes, which needs to be preserved. The methods currently in use, though, are not particularly well categorized.

3.1 Matrix Factorization-Based Algorithms

Matrix factorization in network embedding represents graph properties (such as pairwise node similarity) in matrix format, which is further factorized to yield node embedding. To retain the network structure, various matrices are created. It includes a vertex-context matrix, transition probability matrix, and modularity matrix. It embeds high-dimensional representations of network nodes into a space of reduced dimension while maintaining the network structure. The input to these categories of algorithms is usually a graph made up of data features that are non-relational as well as high-dimensional characteristics. The output consists of a set of node embedding for each respective node.

3.2 Deep Learning-Based Algorithms

Deep learning techniques have excelled in varied research domains, including computer vision, language processing, and others (Cao et al., 2016). In this technique of graph embedding, deep learning models are used for processing graphs. The models created are either straightforward adaptations of other disciplines or brand-new neural network models created exclusively for graph data embedding. The input to these models is in the form of nodes, edges, sampled paths, or the whole graph. The trained models based on Deep Neural Networks can then be used to create embedding of the network. The intermediate values, as well as the final output of the model, are the vectors representing the embedding. These deep learning models with a specially configured objective function preserve the network's proximity and structure. Further, this embedding technique is categorized into the following forms depending on whether a random walk is used or not for generating sample paths in the graph. The deep learning approach provides an efficient way to preserve the non-linear structure for handling massively complex networks.

3.3 Reconstruction of Edge-Based Optimization

The conventional network embedding technique also includes methods that focus on the edges of the network instead of nodes. It takes into consideration the network's local as well as the global proximity levels for embedding. To represent the edge, a low-dimensional vector needs to be created based on the source and target node embeddings concerning the edge. Embedding Methods on networks assist in obtaining the embedding by optimizing the parameters of objective function that retains numerous graph features. This graph embedding technique focuses on the optimization of objective functions based on edge reconstruction. The aim of the objective function used in this technique is to maximize the probability of reconstruction of edges.

3.4 Graph Kernel

Graph kernel is a popular approach that uses graph similarity. The graph kernel refers to a function corresponding to the graph's feature space responsible for representing the graphs and computing similarities between them. The majority of more productive and efficient graph kernels are built around predetermined structural features. It is an example of an R-convolution kernel. Each network representation consists of a vector generated by the graph kernel, and any two of the graphs are analyzed by checking the inner product of their respective vectors. The R-convolution kernels are a general method to present graph kernels on discrete compound items by iteratively breaking down networks into "atomic" sub-networks and comparing each of their pairs. It uses the comprehensive representation provided by some widely used graph kernels.

3.5 Generative Model

The input characteristics and class labels, conditional based on network parameters, can be used to create a generative model. Latent Dirichlet Allocation (LDA), responsible for interpreting a readable document in the form of subjects and topics acting as a distribution over words, serves as an illustration. Both node embedding and edge embedding can be done using a generative model. The input network is typically a heterogeneous network or a network with auxiliary information since it considers node semantics.

Based on the advancements in the above-mentioned techniques, deep learning-based models have shown better performance among the different network embedding methods (Cai et al., 2018). Deep learning-based embedding models can detect usable representations from complicated graph structures automatically. Examples of such implementations include DeepWalk (Perozzi et al., 2014), node2vec (Grover and Leskovec, 2016), and metapath2vec (Dong et al., 2017). The models mentioned above come under the category of random walk-based models. On the other hand, GCN (Kipf and Welling, 2017), struc2vec (Ribeiro et al., 2017), and SDNE (Wang et al., 2016) are some of the deep learning models without using random walk. They can handle sub-graphs of variable size in homogenous graphs and the interactions among nodes in heterogeneous graphs. However, Deep Learning is not without its drawbacks. In Deep Learning using random walks, the knowledge about the global structure is often ignored in favor of a node's local neighbors along the same path. Finding an "ideal" sampling technique is further challenging because embedding and sampling of paths are not simultaneously optimized inside a single framework. The computational cost is often significant for DL without using random walks. The summary of research articles related to different categories of network embedding techniques is provided in Table 1. This tabulation aims to identify and classify them based on their type, paradigm, and tasks for which they can be used. The network embedding paradigm indicates the ability to make predictions based on nodes that are

unknown during the training of the model. Transductive embedding techniques lack the ability to utilize unseen nodes while predicting the outcome. On the other hand, inductive models provide data about the nodes neighboring every node of the network to a symmetric aggregator, which uses the same set of parameters throughout the entire network. Consequently, they can forecast links on hidden nodes that were not present during training. Among the techniques mentioned in Table 1, only a few have been used for defect prediction in previous research works (Qu et al., 2021). As the context of our review work is software defect prediction, which deals with homogeneous networks (where all nodes of the network represent software modules and edges represent the connections between them) (Xie et al., 2021), the techniques listed in Table 1consist of homogeneous embedding techniques. Heterogeneous embedding techniques are not within the scope of our review.

This review work analyzes the papers published for network embedding and network analysis, considering their role in software defect prediction approaches. Fig. 4 depicts a year-by-year distribution of the papers published in the past 15 years for SDP using network analysis and network embedding. Researchers have been exploring the use of network analysis in SDP for quite some time. Zimmermann & Nagappan (2008) performed pioneering work in the area of SDP using network analysis in 2008. The interest of researchers was drawn towards network embedding with the development of advanced embedding techniques in recent years. From 2008 to 2022, network embedding studies have continuously received much attention for developing models that can effectively and efficiently produce reliable defect prediction results. We can conclude from the year-by-year distribution that researchers are very interested in network embedding techniques to develop consistently accurate models.

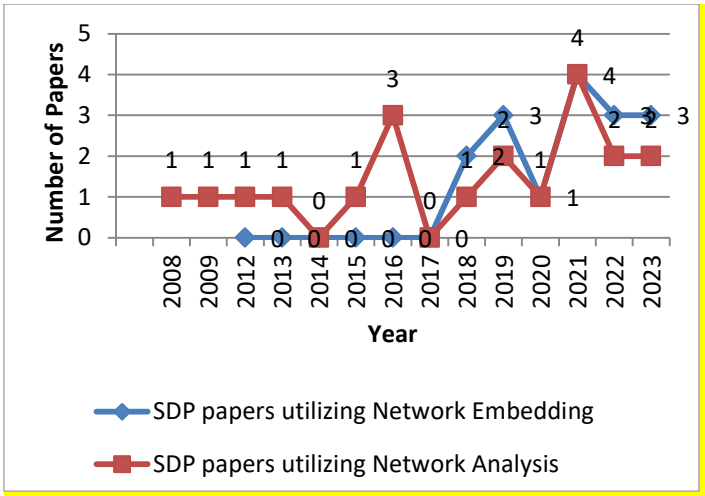


Fig. 4 Year-wise papers published for software defect prediction using network embedding

Table 1 List of network embedding techniques

Method	Category	Publication	Year	Tasks	Type	Paradigm
DeepWalk (Perozzi et al., 2014)	Deep Learning (Using Random Walk)	KDD	2014	Node Classification	Supervised	Transductive
GraRep (Cao et al., 2015)	Node Proximity Matrix Factorization	CIKM	2015	Graph Clustering, Node Clustering, Node Classification, Visualization	Supervised	Transductive
LINE (Tang et al., 2015)	Edge Reconstruction	WWW	2015	Visualization, Node Classification, Link Prediction	Supervised	Transductive
node2vec (Grover and Leskovec, 2016)	Deep Learning (Using Random Walk)	KDD	2016	Node Classification, Link Prediction	Supervised	Transductive
SDNE (Wang et al., 2016)	Deep Learning (Without Using Random Walk)	KDD	2016	Node Classification, Link Prediction, Visualization	Semi-Supervised	Transductive

DNGR (Cao et al., 2016)	Deep Learning (Without Using Random Walk)	AAAI	2016	Graph Clustering, Node Clustering, Node Classification	Unsupervised, Supervised	Transductive
Planetoid (Yang et al., 2016)	Semi-Supervised Learning	JMLR	2016	Node Classification	Semi-Supervised	Inductive, Transductive
HOPE (Ou et al., 2016)	Matrix Factorization	KDD	2016	Edge Reconstruction, Link Prediction, Vertex Recommendation.	Supervised	Transductive
GCN (Kipf and Welling, 2017	Graph-Laplacian Regularization	ICLR	2017	Node classification	Semi-supervised	Transductive
Walklets (Perozzi et al., 2016)	Deep Learning (Using Random Walk)	ASONAM	2016	Graph Clustering, Node Clustering, Node Classification	Unsupervised	Transductive
Graph2vec (Narayana et al., 2017)	Deep Learning (Using Random Walk)	ACM	2017	Graph Clustering, Node Clustering, Node Classification	Unsupervised	Transductive
GraphSAGE (Hamilton et al., 2017)	Deep Learning (Using Random Walk)	NIPS	2017	Node Classification	Supervised, Unsupervised	Inductive
DNE (Shen et al., 2018)	Discrete Matrix Factorization	IJCAI	2018	Node Classification	Supervised	Transductive
NetMF (Qiu et al., 2018)	Matrix Factorization	ACM	2018	Node Classification	Supervised	Transductive
ProNE (Zhang et al., 2019)	Sparse Matrix Factorization	IJCAI	2019	Node Classification	Supervised	Transductive
BinaryNE (Zhang et al., 2021)	Deep Learning (Using Random Walk)	ACM	2021	Node Classification	Supervised	Transductive

4 Research Methodology

4.1 Search Strategy

Our objective is to investigate and evaluate the current patterns and approaches in predicting software defects using network embedding. We specifically focused on research articles published from 2008 to 2022. We conducted a thorough search across multiple databases including IEEE Explorer, ACM Digital Library, Springer, and Elsevier, and also included conference proceedings. Although our search yielded numerous research publications, we carefully selected 70 articles for in-depth analysis based on their title, abstract, citations, and relevance to network embedding and software defect prediction. Prior to composing this review, we conducted a comprehensive review of all the selected research articles.

Inclusion Exclusion Criteria: Our objective was to conduct a comprehensive review of the research trend in software defect prediction using network embedding. To achieve this, we carefully selected 70 papers from the years 2008 to 2022 in order to encompass a wide range of relevant research. Our focus was on papers that evaluated and discussed the performance of models utilizing network embedding techniques.

During our analysis of the published articles, we observed that the initial research efforts primarily revolved around the utilization of network analysis techniques for defect prediction. Notably, Zimmermann and Nagappan made significant contributions to the field in 2008. As network embedding techniques advanced, researchers began exploring their application specifically in predicting fault-prone modules within software systems.

While conducting our review, we encountered an empirical study conducted by Qu and Yin (2021) that explored only seven embedding algorithms. In our work, we aimed to provide a broader understanding of various embedding techniques that could be employed for defect prediction. Consequently, studies that focused on network embedding frameworks and methods unrelated to software defect prediction were excluded from our review. Additionally, we excluded research articles that lacked robust validation procedures or failed to provide experimental findings.

4.2 Research Questions:

In order to maintain a focused approach, we formulated three research questions that would provide valuable insights to researchers interested in studying and utilizing network embedding for software defect prediction. The benefits associated with each research question are outlined in Table 2 below:

Table 2 Motivation for exploring the research questions

Id	Research Question	Motivation
RQ1	How can the accuracy of defect prediction be increased by using different network representations of software projects?	It will help to establish the importance and role of network representation learning in software defect prediction.
RQ2	Which is the best methodology to be employed for network analysis in the context of SDP?	It will be easier for new researchers to understand the existing network analysis techniques prevalent in software defect prediction and replicate previous studies.
RQ3	What are the best techniques utilized for network embedding for predicting software defects?	It will help researchers to identify the research trends and identify techniques that should be researched further.

RQ1 - How can the accuracy of defect prediction be increased by using different network representations of software projects?
This RQ looks into the role played by software dependency networks in the process of predicting software defects. A summary of the papers published is tabulated in Table 2.

RQ2 - Which is the best methodology to be employed for network analysis in the context of SDP?
This RQ focuses on the network analysis techniques useful in SDP. Table 3 provides the list of frequently used network measures. Section 5.2 provides details of the same. A summary of the papers published is tabulated in Table 5.
RQ3 - What are the best techniques utilized for network embedding for predicting software defects?
Table 1 presents a comprehensive overview of the available network embedding techniques that can be applied to handle networks. The aim of Research Question (RQ) is to identify the embedding techniques commonly used for SDP. In Section 5.3 of our work, we conduct a detailed analysis specifically addressing this question. We examine the embedding techniques commonly employed in software defect prediction and assess their effectiveness. Furthermore, we focus on identifying the most promising techniques that have the potential to enhance the predictive capabilities of defect prediction models.

Table 3 List of network measures used frequently by many studies

S. No.	Metric	Definition
1.	Pairs	Total count of distinct node pairs
2.	Ties	edge count represents the total count of directed ties
3.	Size	Count of nodes to which the ego is immediately linked
4.	Density	The proportion of potential ties available currently
5.	nWeakComp	Total weak components / size
6.	ReachEfficiency	2StepReach / size
7.	2StepReach	The proportion of nodes that are present after two steps
8.	EgoBetween	The proportion of routes with the shortest distance between neighbors that go through ego.
9.	Broker	Total pair of nodes that aren't linked directly.
10	nBroker	Broker / size
11.	Betweenness	Determines the count of shortest paths existing between all the other entities
12.	Reachability	Nodes accessible from a specific node
13.	Efficiency	Network's effective size / network's total size
14.	Hierarchy	The distribution of the constraint metrics all over the neighbours
15.	Degree	Total count of nodes next to a specific node
16.	Closeness	The total length of all shortest routes from a specific node to all the remaining nodes
17.	Constraint	Measures the degree of a node's constraints
18.	Power	Specifies the number of links a node has in its neighborhood
19.	Eigenvector	Assigns the nodes of the dependency graph with relative scores

5 Review of different approaches for network analysis and network embedding in software defect prediction

The following section highlights significant research articles on defect prediction using network embedding.

5.1 Significance of network representation of software in Software Defect Prediction

Static code metrics, frequently utilized in defect prediction, focus on collecting statistics about software programs. However, they fail to capture the semantics and structural characteristics of software program modules. This limitation negatively impacts the performance of defect prediction models. To overcome this limitation, various network representations, such as dependency graphs, call graphs, control flow graphs, data flow graphs, and class hierarchy graphs, can be used to effectively capture different aspects of the software system.

Dependency networks in software systems can be represented as graphs, where modules serve as nodes and the connections between modules represent dependencies. If module X calls module Y, module X inherits from module Y, or module X references a variable in module Y, then module X has a dependency on module Y. Each module belongs to both a global network and an ego network. The ego network of a module consists of the module itself and any other modules it depends on or is dependent upon. On the other hand, the global network represents the overall network of all modules within the software system (Tosun et al., 2009).

Zimmermann & Nagappan (2008) conducted significant research in the field of software defect prediction using dependency networks, as documented in their work. They focused on exploring the potential of leveraging module dependencies within software systems. While previous studies had also examined dependencies and faults, Zimmermann et al. contributed to the domain by approaching the issue from a fresh perspective. Their investigation specifically emphasized the interconnections among different software modules. To assess the relationship between dependencies and defect prediction for binaries, they conducted an evaluation using Windows Server 2003. They employed network analysis on dependency graphs to identify the most critical binaries within the system. This analysis required determining dependencies, complexity metrics, and network analysis metrics for Windows Server 2003. Their significant findings highlighted that network measures exhibited higher quality in defect prediction compared to complexity measures. These network measures proved effective in identifying critical binaries that were overlooked by complexity metrics. The majority of network metrics demonstrated noticeable correlations with defects, with most of them being mildly positive. Based on these observations, Zimmermann et al. employed two algorithms, linear regression and logistic regression, to classify the binaries in Windows Server 2003 as either fault-prone or non-fault-prone. The results showed a higher recall value, surpassing complexity metrics by 0.10.

In their study on software, Ma et al. (2016) utilized file-level dependency graphs to explore the relationship between effort-aware defect prediction. They conducted empirical research to examine how network measures influence the prediction of fault-prone modules, taking into account the effort required. The results revealed that several network measures exhibited a significant positive correlation with fault-proneness. Additionally, they observed that the performance of network measures varied depending on the prediction setup and that different projects exhibited inconsistent effects of network measures.

In their paper, Gong et al. (2021) conducted a review on the utilization of dependency networks and the impact of network measures on SDP. They advocated for the integration of network metrics alongside code metrics for more accurate predictions. Additionally, they recommended considering both ego and global metrics as they exhibit distinct behaviors in the prediction process. Yang et al. (2022) developed a sequence of method calls that preserve the structure-related code context information and the semantic details represented by the token sequence. This allows for a deeper investigation of the connections between method call sequences, enabling the learning of both the syntactic structure and the code semantics within the methods. In the study done by Xu et al. (2021) the issue with relying on software metrics to predict all types of defect is highlited. To resolve the issues, they introduced Augmented-CPG, a novel code graph representation method, along with a graph neural network, to predict defects occurring due to data validation issues and user session errors.

Similarly, Gao et al. (2019) performed an analysis on traditional metrics, merged metrics, and network measures. They discouraged the merging of metrics and emphasized the advantages of utilizing complex

network features in software defect prediction across different versions. Table 4 provides a summary of the aforementioned studies.

5.2 Technique used for network analysis in software defect prediction

In recent years, K-core decomposition has been employed by researchers to analyze various types of networks, including protein interaction networks, complex social networks, and large-scale software engineering networks (Yang et al., 2016). A recent application of K-core decomposition involves predicting defects in software through the analysis of Class Dependency Networks (CDNs) (Qu et al., 2021). The researchers discovered an interesting pattern in the defect-prone sections of the software, noting that classes or modules with higher values of k in the k-cores are more likely to have defects. To leverage this insight, they introduced a simple yet effective approach called "top-core". It reorders the classes present in the list of suspicious classes based on their k-core values. This means that classes with higher k values are prioritized and placed higher in the list after reordering. The experiment employed Random Forest and Logistic Regression as the prediction models. The bug prediction results showed an improvement of 11.5 percent for Random Forest and 12.6 percent for Logistic Regression.

In the study presented in (Qu et al., 2021), the software network analysis did not account for the direction and weight of the couplings, thereby overlooking the importance of link strength. Furthermore, the study failed to consider several significant couplings among classes, which has a considerable impact on accurately representing the complex topology of software projects. Consequently, this limitation affects the reliability of metrics derived from the network, such as coreness (Pan et al., 2022). Building upon this research, Du et al. (2022) addressed these shortcomings by incorporating a broader range of coupling types. They proposed CoreBug, an enhanced bug prediction method that takes into consideration effort-related factors. It provides a more precise characterization of classes and the relation between them. CoreBug incorporates nine different types of couplings, considering their direction and strength as well.

Table 4 Summary of papers published using network representation of software for Software Defect Prediction

Authors and Year	Title	Results	Limitations
Zimmermann and Nagappan (2008)	Predicting defects using network analysis on dependency graphs	Network measures have more defect prediction quality as compared to complexity measures. These network measures can find the critical binaries missed by the complexity metrics.	Assessing software with consideration for alternative programming languages.
Ma et al. (2016)	Empirical analysis of network measures for effort-aware fault-proneness prediction	The majority of the network measures have been found to have a significant positive relation to fault-proneness; network measures show performance variation depending on prediction settings, and network measures have inconstant effects on different projects.	Network measures alone inadequately represent intricate software relationships. Effort estimation overlooks testing efficiency.
Gao et al. (2019)	Empirical Study: Are Complex Network Features Suitable for Cross-Version Software Defect Prediction?	Highlighted the benefits of the network's complex features in cross-version software defect prediction.	Does not thoroughly assess the cost parameters associated with the model's utilization.
Gong et al. (2021)	Revisiting the Impact of Dependency Network Metrics on Software Defect Prediction	Supported the usage of network metrics along with code metrics of the software.	Code quality assessment was incomplete due to the limited use of code metrics, resulting in insufficient coverage.
Xu et al. (2021)	Software Defect Prediction for Specific Defect Types based on Augmented Code Graph Representation	Introduced Augmented-CPG, a novel code graph representation method along with graph neural network to predict three different defect types.	The evaluation does not extensively examine the cost parameters linked to the model's usage.
Yang et al. (2022)	Fine-Grained Software Defect Prediction Based on the Method-Call Sequence	Developed a sequence of method calls that preserves both the structural information of the code context and the semantic information represented	Evaluating the model's generalization performance by employing a broader range of classifiers for

		by the token sequence.	validation.

CoreBug introduces a weighted directed class dependency network (WDCDN) as its initial step. It then utilizes the k-core decomposition's generalized form to determine each class's respective coreness value within the WDCDN. By combining the coreness value with the relative risk obtained from logistic regression analysis, CoreBug calculates the likelihood of a given class containing bugs. Experimental results confirmed the superior performance of CoreBug compared to the baseline methods. For more detailed information, please refer to Table 5, which provides an overview of the papers mentioned.

Table 5 Summary of papers published on network analysis techniques in Software Defect Prediction

Authors and Year	Title	Results	Limitations
Qu et al. (2021)	Using K-core Decomposition on Class Dependency Networks to Improve Bug Prediction Model's Practical Performance	There is a greater chance that the classes or modules in k-cores having higher k numbers will likely have defects.	The class dependency network is deficient in encompassing more intricate dependencies.
Du et al. (2022)	CoreBug: Improving Effort-Aware Bug Prediction in Software Systems Using Generalized k-Core Decomposition in Class Dependency Networks	Experimental results on CoreBug confirmed its superiority among the baseline methods.	Needs evaluation spanning a broad spectrum of projects, incorporating those originating from diverse programming languages.

5.3 Most often used network embedding techniques for Software Defect Prediction

In their work, Qu et al. (2018) introduced a novel approach called node2defect, aimed at enhancing the predictive capabilities of defect-predicting algorithms. The key component of this approach is the utilization of node2vec, a recently developed network embedding technique that utilizes random walks. By leveraging node2vec, node2defect enables the automatic extraction of structural aspects from a software's class dependency network. This network embedding technique effectively captures and encodes the dependencies between classes into lower-dimensional vectors, offering new possibilities in defect prediction. Moreover, node2defect integrates the learned structural vectors with conventional software engineering metrics to improve the predictability of faulty modules in software. The proposed method was evaluated through experiments and studies conducted on open-source Java projects. The most significant research finding of the study demonstrated a 9.15% improvement in the F-measure, highlighting the efficacy of the node2defect approach.

Fan et al. (2019) introduced a technique namely Defect Prediction via Attention Mechanism (DP-AM) that leverages program semantics and static metrics using the attention mechanism. DP-AM utilizes abstract syntax trees (ASTs) of programs to generate vectors, which are further encoded in the form of digital vectors through mapping and the word embedding techniques. These numerical vectors are then fed into a Recurrent Neural Network (RNN) to automatically learn the semantic aspects of software programs. The framework employs a self-attention process to establish connections between these features. Finally, DP-AM combines these semantic features with conventional static measurements to achieve accurate software fault prediction. Experiments conducted on Java projects demonstrated the effectiveness of DP-AM, resulting in an average improvement of 11% in the F1 measure. This showcases the capability of DP-AM to enhance software defect prediction by effectively utilizing program semantics and static metrics.

Tong et al. (2019) conducted significant research in software defect prediction using network embedding. They employed the kernel spectral embedding technique (KSETE) specifically in the context of cross-project defect prediction. Through their experiments, they demonstrated the effectiveness of KSETE in accurately predicting defects across different projects.

In another study by Qu and Yin (2021), the capabilities of various network embedding methods in software bug prediction was evaluated. They examined seven different network embedding algorithms, utilizing the node2defect approach which combines traditional software engineering metrics with the embedded vectors. The findings revealed that node2defect outperformed traditional metrics by a significant margin of +14.64% in terms of the MCC (Matthews Correlation Coefficient) score. The network embedding

algorithm, ProNE, demonstrated superior performance compared to the other network embedding algorithms. Furthermore, Figure 5 illustrates the average execution time of the seven algorithms used in the study (Qu and Yin, 2021), with ProNE exhibiting the fastest execution time of 1.14 seconds.

Zeng et al. (2021) presented a model that utilizes the structural features of software by employing a class dependency network and generating embeddings. To address data imbalance in the defect dataset, they employed SMOTETomek sampling technique. This approach was evaluated on eight open-source programs, and the final outcomes showcased the superiority of GCN2defect over the leading method. On average, GCN2defect outperformed the leading method by 6.84% to 23.85% F-measure value. This demonstrates the effectiveness of the model in improving software defect prediction accuracy.

In their research, Yang et al. (2023) examined the relationships between methods by leveraging a method-calling network, which effectively captures the structural details of a software system. Their proposed framework comprises two distinct stages. In the first stage, network metrics are extracted from the network of method calls, and metrics from network embedding are generated using the node2vec embedding technique. In the second stage, the extracted network metrics and embedding metrics are integrated with traditional code metrics to be the input for the defect prediction model.

A deep learning-based approach was introduced by Tang et al. (2023) to detect defects at the function-level. The method utilizes a control flow graph to obtain semantic features of the code as node embeddings. Additionally, graph neural networks are employed to obtain structured data from the graph. Experimental results indicate that the proposed method, called CSGVD, achieves an accuracy of 64.46% on a dataset gathered from CodeXGLUE for defect detection.

A refined level defect detection framework was developed by Dong et al. (2023), incorporating graph embeddings at the subgraph level. The framework utilized the Code Property Graph to extract syntactic and semantic knowledge from the program code, enabling the creation of subgraphs. Each subgraph is then embedded, and machine learning classifiers are used to evaluate the datsets originating from C/C++ projects in NVD and SARD. The framework achieves an impressive 95.15% F1-measure, validating the effectiveness of this approach.

Table 6 provides a summary of the papers mentioned above, showcasing their key contributions and findings. Moreover, based on an extensive study of various network embedding techniques, Table 7 presents a compilation of the most promising techniques that can enhance the predictive capability of defect prediction models.

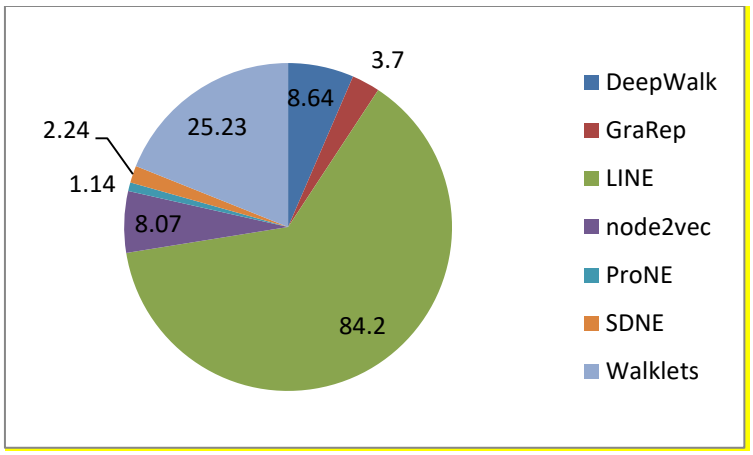


Fig. 5 Average execution time of network embedding algorithms as per Qu & Yin (2021)

As mentioned previously, both the embedding methods mentioned earlier and those utilized in (Qu & Yin, 2021) are transductive in nature. This implies that the embeddings are learned based on the existing graph structure, restricting predictions to instances that have already been observed in the network during training. However, given that the software undergoes regular structural changes such as module additions or removals, it may be more advantageous

Table 6 Summary of papers published using Network Embedding techniques for Software Defect Prediction

Authors and Year	Title	Results	Limitations

Qu et al. (2018)	Node2defect: using network embedding to improve software defect prediction	Presented a completely new approach called node2defect to improve the prediction capability of defect-predicting algorithms using node2vec. It helps to automatically extract the structural aspects of the software using its class dependency network.	The embedding generated from the class dependency network inadequately captures the relationships among software components.
Fan et al. (2019)	Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction	Proposed a framework to fully use program semantics and static metrics using the Attention Mechanism namely defect Prediction via Attention Mechanism (DP-AM). It uses the program's abstract syntax trees and word embeddings.	Overlooks the significance of embedding dimension effects. Word embeddings struggle to adequately capture code-specific semantics, structures, and context.
Tong et al. (2019)	Kernel Spectral Embedding Transfer Ensemble for Heterogeneous Defect Prediction	They successfully employed the kernel spectral embedding technique (KSETE) in cross-project defect prediction.	The model is evaluated using a limited selection of projects.
Qu et al. (2021)	Evaluating network embedding techniques' performances in software bug prediction.	Evaluated the performance of network embedding techniques in the prediction of software bugs. The study compared the performance of seven network embedding algorithms by utilizing node2defect.	Does not incorporate more advanced dependency capturing graphs and embedding techniques.
Zeng et al.(2021)	GCN2defect : Graph Convolutional Networks for SMOTETomek-based Software Defect Prediction	The proposed model involves obtaining structural features of software using a class dependency network and generating embeddings. The model was evaluated on eight open-source projects, and it achieved improvements in terms of F-measure.	The evaluation is deficient in terms of a broad spectrum of projects, and there is a need to expand the assessment metrics.
Yang et al.(2023)	A Method-Level Defect Prediction Approach Based on Structural Features of Method-Calling Network	Presented a new framework for automatically encoding method calling networks with node2vec and then combine the resulting embeddings with network metrics of the method calling network.	The model's capacity to capture complex patterns hasn't been evaluated for different embedding dimensions, a crucial factor that directly influences its performance.
Tang et al. (2023)	CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection	The method utilizes a control flow graph to extract semantic features of the code as node embeddings. Additionally, graph neural networks are employed to extract structured data from the graph.	Estimating computational expenses and conducting comparisons with baseline measures.
Dong et al. (2023)	SedSVD: Statement-level software vulnerability detection based on Relational Graph Convolutional Network with subgraph embedding	The framework utilized the Code Property Graph to extract semantic and syntactic knowledge from the source code, enabling the creation of subgraphs. Each subgraph is then embedded, and machine learning classifiers are used to evaluate the datsets originating from C/C++ projects in NVD and SARD.	Lacks evaluation that spans projects involving a diverse range of programming languages.

to employ an inductive method that enables predictions to be made on occurrences that have not been observed in the network during training. Surprisingly, no research has been published utilizing the inductive network embedding method for software defect prediction. According to Table 1, network embedding techniques like Planetoid (Yang et al., 2016) and GraphSAGE (Hamilton et al., 2017) can be employed to predict fault-prone modules using the inductive approach.

Table 7 Summary of enhanced network embedding techniques.

Method	Technique	Advantages

DNE	Discrete Matrix Factorization	While achieving competitive classification results, DNE has less computational and storage complexity than leading network embedding techniques.
NetMF	Matrix Factorization	NetMF's direct factorization consistently outperforms implicit approximation models. It combines LINE, PTE, DeepWalk, and node2vec in a matrix factorization framework.
BinaryNE	Deep Learning Based, With Random Walk	BinaryNE not only outperforms traditional continuous vector-based network embedding methods in search speed by more than 23 times but also in terms of search quality.

6 Discussion

The previous section highlights the significant influence of network analysis on the outcomes of defect prediction models. The effectiveness of these models heavily relies on the utilization of software dependency networks and network embedding techniques, which have proven to be highly advantageous in bug prediction. Over time, researchers have transitioned from static software metrics to object-oriented metrics and, subsequently, to incorporating module dependencies within the software to improve the accuracy of defect prediction. Static metrics proved inadequate in capturing the intricate and dynamic nature of modern object-oriented software, leading to the adoption of object-oriented metrics for defect prediction. However, these metrics faced limitations in providing the defect prediction models with the necessary generalization capabilities, as the metrics data relied on the specific tools used for their generation. To address this issue, network analysis emerges as a viable solution, offering a means to visualize and analyze complex networks of significant magnitude. Moreover, software dependency networks enable the capture of the structural dependencies among software modules. The application of K-core decomposition to class dependency networks (referred to as CoreBug) enhances the effectiveness of effort-aware bug prediction models by accurately characterizing classes and the relationships between them. Our survey findings also indicate that the majority of network measures display a noteworthy positive correlation with fault-proneness. This supports the notion of combining network metrics with traditional software engineering metrics. It is worth noting that network embedding methodologies can effectively capture and encode class dependencies into lower-dimensional vectors. Among the techniques discussed in Section 5.3, ProNE stands out as the fastest approach. An interesting observation from our results is that DNE, NetMF, and BinaryNE are techniques that can potentially enhance defect prediction accuracy. Furthermore, we observe that the network embedding techniques employed in software defect prediction are predominantly transductive in nature. However, considering the regular structural changes that occur in software, it is worth exploring the use of inductive network embedding methods to determine if they can contribute to improved prediction accuracy.

6.1 Challenges and Future Research Directions

6.1.1 Enhancing Generalizability and Scalability of SDP models

Existing SDP approaches encompass a software project represented as a network of modules, where nodes denote entities like classes and packages, and edges depict the relationships between them. The network embeddings derived from these graphs capture intricate information pertaining to the modules in the network structure. In past studies focusing on software metrics, the prevalent approach often revolved around adapting models to the unique attributes of a particular software or utilizing a generic model customized for a specific software and context. This practice led to the development of numerous models, each tailored to a distinct software type. Despite the flexibility of embedding-based models, which can leverage embeddings from diverse input graphs, they have struggled to attain the desired level of generality expected from a comprehensive graph model for defect prediction. Consequently, a promising direction for future research lies in investigating whether a model with the capability to effectively learn from all types of software graphs exists, thereby addressing the current limitations and enhancing the versatility of defect prediction methodologies.

6.1.2 Beyond Classification: Explainability in SDP models

Applying eXplainable Artificial Intelligence (XAI) techniques in predicting software defects not only improves the transparency and dependability of the models but also empowers developers to make well-informed decisions and implement targeted actions for defect mitigation. Furthermore, it illuminates how embeddings impact the classification mechanism of modules as defect-prone or non-defect-prone, shifting

from perceiving the Software Defect Prediction (SDP) model as a black box. Understanding the key embeddings that significantly influence the model's decisions assists developers and stakeholders in comprehending the factors contributing to software defect predictions. This proves beneficial by providing insights into why a specific module or code segment is identified as more susceptible to defects. Advancements in methodologies, such as attention mechanisms, offer the capability to tackle the non-interpretable aspects of SDP models, particularly in the case of deep learning models that depend on network embeddings from graphs. The obtained attention weights possess the ability to highlight the sections of the input graph that have a more significant impact on defect prediction results, as indicated by their higher attention scores.

6.1.3 Mitigating Bias in SDP Models

Existing SDP models that rely on embeddings have not investigated the impact of balancing techniques. The absence of such considerations may result in biased models, introducing unfairness in defect prediction quality. The underlying challenge in SDP addresses the classification of defect-prone and non-defect-prone classes. As non-defect-prone classes typically outnumber defect-prone ones, the embedding dataset exhibits an imbalance, affecting the development of models. While previous research has attempted to mitigate this bias through the application of oversampling and undersampling techniques, these methods have primarily been applied to software metrics datasets, rather than embedding datasets. In particular, this gap highlights the necessity of addressing the unfairness issue arising from data imbalance through an empirical study focused on embedding datasets sourced from diverse projects. Another contributing factor to this bias is the utilization of inadequate evaluation metrics, the insufficiency of which tilts the focus of learning objectives towards accuracy of defect prediction. While recent studies have attempted to mitigate this bias by assessing developed models using parameters beyond accuracy, the evaluation of related embedding-based studies in the domain of defect prediction research remains limited, with numerous issues yet to be thoroughly investigated. For example, it is essential to investigate whether the learning process of embeddings from graphs influences the prediction results. Additionally, exploring the potential existence of algorithmic bias among various network embedding techniques and examining the interplay between fairness, bias, discrimination, and the accuracy of embeddings-based SDP models requires further research.

6.1.4 Validating SDP models: Establishing Links with Practical Implementations

The majority of established Software Defect Prediction (SDP) models rely on a commonly utilized dataset widely employed by researchers in the field. This dataset encompasses defect data from sources such as the PROMISE repository, AEEM, NASA datasets, SOFTLAB Dataset, and the ReLink Dataset. The extended usage of these datasets over time has created a gap, posing a challenge for applying models developed with them to the complexities of present-day software. Given the diverse nature of modern software projects, it is crucial to approach the issue of Software Defect Prediction (SDP) in a manner that takes into account the varied characteristics of these projects. Consequently, it is crucial to subject the developed models to meticulous and thorough validation processes before incorporating them into real-world applications. Hence, bridging this gap necessitates collaborative initiatives that engage diverse industries and research communities. These efforts aim to comprehend the functionality and constraints of models in practical projects, integrating methodologies to overcome model limitations based on this understanding. Developing models grounded in new benchmark datasets is crucial for assessing their applicability in diverse industries, especially within the context of modern complex software systems.

6.1.5 Enhancing Robustness in SDP models

It is also imperative to ensure that developed models are resilient to changes in software and programming paradigms. Performing a sensitivity analysis on the model's performance with regard to data and programming paradigms provides valuable insights into its stability. This analysis aids in identifying potential improvements in model specifications, design, and training processes to enhance the practical applicability of the model in real-world scenarios. Beyond its predictive functions, measuring the uncertainty of a model proves to be a crucial asset for end-users. This method affords users the ability to finely adjust and appraise the confidence associated with the model's predictions. Gaining insight into the degree of uncertainty offers a nuanced perspective, enabling individuals to make more informed decisions grounded in the reliability and certainty of the model's output. This not only elevates the overall

effectiveness of the model but also fosters transparency and accountability in the decision-making process. Assessing how the outcomes of models used for defect prediction are influenced by the size of embeddings is a crucial aspect of fine-tuning model performance. Embedding size, in this context, refers to the dimensionality of the vector space where features are represented. It is tied to how well the model captures and represents patterns and relationships in the data. To thoroughly examine this influence, one should systematically explore different embedding sizes, such as 32, 64, 128, and 256. Maintaining a consistent dataset and model architecture across experiments while varying only the embedding size is essential. The use of cross-validation ensures result robustness, enabling a comprehensive analysis of model performance.

6.1.6 Prediction on Dynamic graphs

The majority of studies leveraging network embeddings for defect prediction have used static graphs to derive embeddings for model construction, thus overlooking the dynamic nature of the software development process. In practical situations, software modules are subject to continuous alterations, involving instances where modules are either removed or new ones are added during the maintenance phase. Additionally, dynamic changes occur when introducing or removing new feature updates in the software. Therefore, for effectively capturing these dynamic variations, it is recommended to employ appropriate graph structures, such as dynamic call graphs. This strategy equips the model to adapt to modifications in the software, ensuring a more responsive and adaptable predictive framework. An alternative approach entails utilizing a dynamic graph presented as a series of graph snapshots, with a specific emphasis on capturing software changes. This technique facilitates a detailed examination of the evolving structure, contributing to a thorough understanding of the software's dynamic nature over time. Integrating dynamic random walk sampling with other algorithms that capture graph's dynamic properties can be used to address the concern. This integrated strategy harnesses the strengths of multiple algorithms, offering a more comprehensive and nuanced understanding of evolving graph structures in dynamic environments. Pursuing this direction in research is deemed crucial.

6.2 Threats to Validity

The following section examines the significant factors that can have effects on the validity of our study.

6.2.1 Internal Validity

The defect prediction strategies examined in the research papers we surveyed primarily adopt a transductive approach, which may introduce biases in the obtained results. To address this potential limitation, we conducted a comparative study of inductive embedding techniques that can be employed to develop defect prediction models, aiming to provide a more comprehensive and unbiased perspective. Another significant internal validity concern is that our work solely focuses on evaluating the performance of network embedding in comparison to traditional code metrics for software defect prediction. We do not consider other categories of metrics such as change metrics, churn metrics, etc. This limitation arises due to the unavailability of software project histories, leaving us with only the source code as the available data. Nonetheless, we believe that our study results will support researchers in making better use of the software source code and contribute to their understanding and application of defect prediction methodologies.

6.2.2 External Validity

In our study, a potential threat to external validity pertains to the data sets utilized. The majority of papers included in our analysis relied on experiments conducted on public data sets, while only a few utilized private data sets. These data sets exhibit variations in terms of software metric sets, dimensions, and types. Consequently, the generalizability of the results to software projects that are not open source may be compromised. To mitigate this potential threat, it would be beneficial to consider a wider range of software projects of commercial nature during future investigations. By including such projects in the analysis, the outcomes can be improved, and the applicability of the results can be expanded to a broader context.

7 Conclusion

In a pioneering effort, this study presents a comprehensive review and analysis of the literature, focusing on works utilizing network embedding for Software Defect Prediction. It explores a wide range of embedding techniques employed in this domain. The study initially presents a formal description of network analysis in software and network embedding. The research meticulously analyzes papers from two distinct perspectives:

the embedding methods and the specific developments in its application to SDP. The analysis focuses on three major groups of methods: deep learning-based, factorization-based, and random walk-based, examining the structure and features extracted by various network embedding techniques. The article also emphasizes the application of network analysis in dependency graphs and highlights recent advancements in network embedding techniques used for SDP. We highlight the advantages of the utilization of enhanced embedding techniques, as they have the potential to increase defect prediction models' efficiency. By focusing on these aspects, the study provides a holistic understanding of the diverse range of techniques employed in the crucial task of determining whether a software module is susceptible to defects. The existing works utilize network embedding techniques following a transductive paradigm. This study emphasizes the need to explore the performance of SDP techniques in the inductive paradigm, as they can more effectively capture the nature of software development. Furthermore, this study goes beyond the current state of knowledge to identify the challenges inherent in this domain. Recognizing the complexities and gaps in existing methodologies, the research aims to shed light on areas where further investigation is essential. By addressing the existing challenges and outlining avenues for future exploration, this work contributes not only to the understanding of embedding techniques usage in SDP but also serves as a roadmap for researchers seeking to enhance the effectiveness of defect prediction in software.

Statements and Declarations

Author's contributions SM and KSP conceived the initial research idea. SM conducted the literature search and data analysis. SM drafted the initial paper and managed the revisions. KSP planned the revisions and made critical adjustments to both the initial and revised manuscripts. PKG supervised the study and provided feedback. All authors have read and approved the final manuscript.

Data availability The data that support the findings of this study are available on request from the corresponding author.

Code availability Not Applicable.

Conflict of interest The authors declare that they have no conflict of interest.

Funding No funding was received for conducting this study.

Ethics approval Not Applicable.

Consent to participate Not Applicable.

Consent for publication Not Applicable.

References

1. Alharthi, Z. S., Alsaeedi, A., & Yafooz, W. M. S. (2021). Software defect prediction approaches: A review. In *Proceedings of the 4th International Conference on Bio-Engineering for Smart Technologies* (pp. 1-6). https://doi.org/10.1109/BioSMART54244.2021.9677869
2. Ali, Z., Qi, G., Muhammad, K., Ali, B., & Abro, W. A. (2020). Paper recommendation based on heterogeneous network embedding. *Knowledge-Based Systems*, 210, 106438. https://doi.org/10.1016/j.knosys.2020.106438
3. Bahaweres, R. B., Jumral, D., Hermadi, I., Suroso, A. I., & Arkeman, Y. (2021). Hybrid software defect prediction based on LSTM (Long Short Term Memory) and word embedding. In *Proceedings of the 2nd International Conference On Smart Cities, Automation & Intelligent Computing Systems* (pp. 70-75). https://doi.org/10.1109/ICON-SONICS53103.2021.9617182
4. Hossain, M., & Chen, H. (2022). Application of Machine Learning on Software Quality Assurance and Testing: A Chronological Survey. *International Journal of Computers and their Applications*, *29*(3), 150-157.
5. Cai, H., Zheng, V., & Chang, K. (2018). A comprehensive survey of graph embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge & Data Engineering*, *30*(9), 1616-1637. https://doi.org/10.1109/TKDE.2018.2807452
6. Cao, S., Lu, W., & Xu, Q. (2015). Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (pp. 891-900). ACM. https://doi.org/10.1145/2806416.2806512

7. Cao, S., Lu, W., & Xu, Q. (2016). Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (pp. 1145-1152). AAAI Press.
8. Chen, H., Su, X., Tian, Y., Perozzi, B., Chen, M., & Skiena, S. (2018). Enhanced network embeddings via exploiting edge labels. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 4 pages). https://doi.org/10.1145/3269206.3269270
9. Chen, L., Ma, W., Zhou, Y., Xu, L., Wang, Z., Chen, Z., & Xu, B. (2016). Empirical analysis of network measures for predicting high severity software faults. *Science China Information Sciences*, 59, Article 122901. https://doi.org/10.1007/s11432-015-5426-3
10. Coscia, J. L. O., Crasso, M., Mateos, C., & Zunino, A. (2012). Estimating Web service interface complexity and quality through conventional object-oriented metrics. In *15th Ibero-American Conference on Software Engineering*. https://doi.org/10.19153/cleiej.16.1.4
11. Coscia, J. L. O., Crasso, M., Mateos, C., Zunino, A., & Misra, S. (2012). Predicting web service maintainability via object-oriented metrics: A statistics-based approach. *Computational Science and Its Applications, Lecture Notes in Computer Science*, 7336. https://doi.org/10.1007/978-3-642-31128-4_3
12. Dai, Q., Shen, X., Zhang, L., Li, Q., & Wang, D. (2019). Adversarial Training Methods for Network Embedding. In *Proceedings of the World Wide Web Conference* (pp. 329-339). https://doi.org/10.1145/3308558.3313445
13. Dong, T., Shi, H., Zhu, Y., Li, K., Chai, F., & Wang, Y. (2019). Embedded software reliability prediction based on software life cycle. In *Proceedings of the IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering* (pp. 725-729). https://doi.org/10.1109/ISKE47853.2019.9170437
14. Dong, Y., Chawla, N. V., & Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 135-144). https://doi.org/10.1145/3097983.3098036
15. Dong, Y., Tang, Y., Cheng, X., Yang, Y., & Wang, S. (2023). SedSVD: Statement-level software vulnerability detection based on Relational Graph Convolutional Network with subgraph embedding. *Information and Software Technology*, 158. https://doi.org/10.1016/j.infsof.2023.107168
16. Du, X., Wang, T., Wang, L., Pan, W., Chai, C., Xu, X., Jiang, B., & Wang, J. (2022). CoreBug: Improving effort-aware bug prediction in software systems using generalized k-core decomposition in class dependency networks. *Axioms*, *11*(5), 205. https://doi.org/10.3390/axioms11050205
17. Du, X., Yan, J., Zhang, R., & Zha, H. (2022). Cross-Network Skip-Gram Embedding for Joint Network Alignment and Link Prediction. *IEEE Transactions on Knowledge and Data Engineering*, *34*(3), 1080-1095. https://doi.org/10.1109/TKDE.2020.2997861
18. Fan, G., Diao, X., Yu, H., Yang, K., & Chen, L. (2019). Deep semantic feature learning with embedded static metrics for software defect prediction. In *Proceedings of the 26th Asia-Pacific Software Engineering Conference* (pp. 244-251). https://doi.org/10.1109/APSEC48747.2019.00041
19. Gao, H., Lu, M., Pan, C., & Xu, B. (2019). Empirical Study: Are complex network features suitable for cross-version software defect prediction? In *Proceedings of the IEEE 10th International Conference on Software Engineering and Service Science* (pp. 1-5). https://doi.org/10.1109/ICSESS47205.2019.9040793
20. Gong, L., Rajbahadur, G. K. K., Hassan, A. E., & Jiang, S. (2021). Revisiting the impact of dependency network metrics on software defect prediction. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2021.3131950
21. Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151, 78-94. https://doi.org/10.1016/j.knosys.2018.03.022
22. Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd International Conference on Knowledge Discovery & Data Mining* (pp. 855-864). https://doi.org/10.1145/2939672.2939754
23. Gurung, S. (2022). Performing software defect prediction using deep learning. *Computer and Information Science*, 1697. Springer. https://doi.org/10.1007/978-3-031-22405-8_25
24. Halstead, M. H. (1977). Elements of software science (Operating and programming systems series).
25. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and Applications. *IEEE Data Engineering, 40*(3), 52-74. arXiv:1709.05584
26. Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Proceedings of the 28th International Conference on Neural Information Processing Systems* (pp. 1025-1035). https://doi.org/10.48550/arXiv.1706.02216
27. Harrison, R., Counsell, S. J., & Nithi, R. V. (1998). An evaluation of the mood set of object-oriented software metrics. *IEEE Transactions on Software Engineering, 24*(6), 491-496. https://doi.org/10.1109/32.689404
28. Hou, M., Ren, J., Zhang, D., Kong, X., Zhang, D., & Xia, F. (2020). Network embedding: Taxonomies, frameworks and applications. *Computer Science Review*, 38, 100296. https://doi.org/10.1016/j.cosrev.2020.100296
29. Huo, X., Yang, Y., Li, M., & Zhan, D. (2018). Learning semantic features for software defect prediction by code comments embedding. In *Proceedings of the IEEE International Conference on Data Mining* (pp. 1049-1054). https://doi.org/10.1109/ICDM.2018.00133
30. Jureczko, M., & Spinellis, D. (2010). Using object-oriented design metrics to predict software defects. *Models and Methods of System Dependability* (pp. 69-81). Oficyna Wydawnicza Politechniki Wrocławskiej.
31. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning* Representations (pp. 1-14). arXiv:1609.02907
32. Li, N., Liu, J., He, Z., Zhang, C., & Xie, J. (2022). Network Embedding with dual generation tasks. *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/TKDE.2022.3187851

33. Li, T., Zhang, J., Yu, P. S., Zhang, Y., & Yan, Y. (2018). Deep dynamic network embedding for link prediction. *IEEE Access*, 6, 29219-29230. https://doi.org/10.1109/ACCESS.2018.2839770
34. Ma, W., Chen, L., Yang, Y., Zhou, Y., & Xu, B. (2016). Empirical analysis of network measures for effort-aware fault-proneness prediction. *Information and Software Technology*, 69, 50-70. https://doi.org/10.1016/j.infsof.2015.09.001
35. McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308-320. https://doi.org/10.1109/TSE.1976.233837
36. Narayana, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., & Jaiswal, S. (2017). graph2vec: Learning distributed representations of graphs. arXiv:1707.05005
37. Nguyen, T. H. D., Adams, B., & Hassan, A. E. (2010). Studying the impact of dependency network measures on software quality. In *Proceedings of the IEEE International Conference on Software Maintenance* (pp. 1-10). https://doi.org/10.1109/ICSM.2010.5609560
38. Ou, M., Cui, P., Pei, J., Zhang, Z., & Zhu, W. (2016). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1105-1114). https://doi.org/10.1145/2939672.2939751
39. Pan, W., Ming, H., Yang, Z., & Wang, T. (2022). Comments on using k-core decomposition on class dependency networks to improve bug prediction model's practical performance. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2022.3140599
40. Pereira, J., Groen, A. K., Stroes, E. S. G., & Levin, E. (2019). Graph space embedding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (pp. 3253-3259). https://doi.org/10.24963/ijcai.2019/451
41. Perozzi, B., Kulkarni, V., & Skiena, S. (2016). Walklets: Multiscale graph embeddings for interpretable network classification. ArXiv:abs/1605.02115.
42. Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge discovery and data mining* (pp. 701-710). https://doi.org/10.1145/2623330.2623732
43. Pinzger, M., Nagappan, N., & Murphy, B. (2008). Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 2-12). https://doi.org/10.1145/1453101.1453105
44. Premraj, R., & Herzig, K. (2011). Network versus code metrics to predict defects: A replication study. In *International Symposium on Empirical Software Engineering and Measurement* (pp. 215-224). https://doi.org/10.1109/ESEM.2011.30
45. Qiu, J., Yuxiao, D., Ma, H., Li, J., Wang, K., & Tang, J. (2018). Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec. *In Proceedings of the 11th ACM Int. Conf. on Web Search and Data Mining* (pp. 459-467). https://doi.org/10.1145/3159652.3159706
46. Qu, Y., Liu, T., Chi, J., Jin, Y., Cui, D., He, A., Zheng, Q. (2018). Node2defect: using network embedding to improve software defect prediction. In *Proceedings of the 33rd ACM/IEEE Int. Conf. on Automated Software Engineering* (pp. 844-849). https://doi.org/10.1145/3238147.3240469
47. Qu, Y., & Yin, H. (2021). Evaluating network embedding techniques' performances in software bug prediction. *Empirical Software Engineering*, 26, 60. https://doi.org/10.1007/s10664-021-09965-5
48. Qu, Y., Zheng, Q., Chi, J., Jin, Y., He, A., Cui, D., Zhang, H., & Liu. (2021). Using K-core Decomposition on Class Dependency Networks to improve bug prediction model's practical performance. *IEEE Transactions on Software Engineering*, 47, 348-366. https://doi.org/10.1109/TSE.2019.2892959
49. Ribeiro, L. F. R., Saverese, P. H., & Figueiredo, D. R. (2017). Struc2vec: Learning node representations from structural identity. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 385-394). https://doi.org/10.1145/3097983.3098061
50. Shen, X., Pan, S., Liu, W., Ong, Y., & Sun, Q. (2018). Discrete network embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 3549-3555).
51. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). LINE: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web* (pp. 1067-1077). https://doi.org/10.1145/2736277.2741093
52. Tang, S., Meng, Z., & Liang, S. (2022). Dynamic Co-Embedding Model for temporal attributed networks. *IEEE Transactions on Neural Networks and Learning Systems*. https://doi.org/10.1109/TNNLS.2022.3193564
53. Tang, W., Tang, M., Ban, M., Zhao, Z., & Feng, M. (2023). CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. Journal of Systems and Software, 199. https://doi.org/10.1016/j.jss.2023.111623
54. Tong, H., Liu, B., & Wang, S. (2019). Kernel spectral embedding transfer ensemble for heterogeneous defect prediction. *IEEE Transactions on Software Engineering*, *47*(9), 1886-1906. https://doi.org/10.1109/TSE.2019.2939303
55. Tosun, A., Turhan, B., & Bener, A. (2009). Validation of network measures as indicators of defective modules in software systems. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering* (pp. 1-9). https://doi.org/10.1145/1540438.1540446
56. Wang, D., Cui, P., & Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd Int. Conf. on Knowledge Discovery and Data Mining* (pp. 1225-1234). https://doi.org/10.1145/2939672.2939753
57. Wang, X., Lu, L., Wang, B., Shang, Y., & Yang, H. (2022). Software defect prediction via GIN with hybrid graphical features. *In IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion*, 411-416. https://doi.org/10.1109/QRS-C57518.2022.00066
58. Wang, Z., Ye, X., Wang, C., Cui, J., & Yu, P. S. (2021). Network embedding with completely-imbalanced labels. *IEEE Transactions on Knowledge and Data Engineering, 33*(11), 3634-3647. https://doi.org/10.1109/TKDE.2020.2971490

59. Xie, Y., Yu, B., Lv, S., Zhang, C., Wang, G., & Gong, G. (2021). A survey on heterogeneous network representation learning. *Pattern Recognition*, 116, 107936. https://doi.org/10.1016/j.patcog.2021.107936
60. Xu, J., Ai, J., & Shi, T. (2021). Software Defect Prediction for Specific Defect Types based on Augmented Code Graph Representation. *In Proceedings of the Conference on Dependable Systems and Their Applications* (pp. 669-678). https://doi.org/10.1109/DSA52907.2021.00097
61. Yang, C., Shi, C., Liu, Z., Tu, C., & Sun, M. (2021). Network Embedding: Theories, methods, and applications. *Springer Cham*.
62. Yang, F., Huang, Y., Xu, H., Xiao, P., & Zheng, W. (2022). Fine-Grained software defect prediction based on the method-call sequence. *Computational Intelligence and Neuroscience*, 4311548. https://doi.org/10.1155/2022/4311548
63. Yang, F., Xu, H., Xiao, P., Zhong, F., & Zeng, G. (2023). A Method-Level defect prediction approach based on structural features of method-calling network. *IEEE Access*, 11, 7933-7946. https://doi.org/10.1109/ACCESS.2023.3239266
64. Yang, Y., Ai, J., & Wang, F. (2018). Defect prediction based on the characteristics of multilayer structure of software network. *In Proceedings of the IEEE International Conference on Software Quality, Reliability and Security Companion* (pp. 27-34). https://doi.org/10.1109/QRS-C.2018.00019
65. Yang, Y., Harman, M., Krinke, J., Islam, S., Binkley, D., Zhou, Y., & Xu, B. (2016). An empirical study on dependence clusters for effort-aware fault-proneness prediction. In *Proceedings of the 31st IEEE/ACM Int. Conf. on Automated Software Engineering* (pp. 296-307).
66. Yang, Z., Cohen, W. W., & Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd Int. Conf. on Int. Conf. on Machine Learning* (pp. 40-48). https://doi.org/10.48550/arXiv.1603.08861
67. Zeng, C., Zhou, C. Y., Lv, S. K., He, P., & Huang, J. (2021). GCN2defect: Graph Convolutional Networks for SMOTETomek-based software defect prediction. In *IEEE 32nd International Symposium on Software Reliability Engineering* (pp. 69-79). https://doi.org/10.1109/ISSRE52982.2021.00020
68. Zhang, D., Yin, J., Zhu, X., & Zhang, C. (2021). Search efficient binary network embedding. *ACM Transactions on Knowledge Discovery and Data*, *15*(4), Article 61, 1-27. https://doi.org/10.1145/3436892
69. Zhang, J., Dong, Y., Wang, Y., Tang, J., & Ding, M. (2019). ProNE: Fast and scalable network representation learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (pp. 4278-4284). https://doi.org/10.24963/ijcai.2019/594
70. Zhu, W., Wang, X., & Cui, P. (2020). Deep Learning for learning graph representations. *W. Pedrycz & S. M. Chen (Eds.), Deep Learning: Concepts and Architectures. Studies in Computational Intelligence*, 866, 99-115. https://doi.org/10.1007/978-3-030-31756-0_6
71. Zimmermann, T., & Nagappan, N. (2008). Predicting defects using network analysis on dependency graphs. In *Proceedings of the ACM/IEEE 30th Int. Conf. on Software Engineering* (pp. 531-540). https://doi.org/10.1145/1368088.1368161