Hindawi
Scientific Programming
Volume 2021, Article ID 2323100, 11 pages
https://doi.org/10.1155/2021/2323100

Research Article

Deep Learning Software Defect Prediction Methods for Cloud Environments Research

Wenjian Liu,[1] Baoping Wang ,[1] and Wennan Wang[1,2]

[1]*Faculty of Data Science, City University of Macau, Macau, China*
[2]*Alibaba Cloud Big Data Application College, Zhuhai College of Science and Technology, Zhuhai, China*

Correspondence should be addressed to Baoping Wang; d19092105076@cityu.mo

Received 2 October 2021; Revised 26 October 2021; Accepted 27 October 2021; Published 18 November 2021

Academic Editor: Tongguang Ni

This paper provides an in-depth study and analysis of software defect prediction methods in a cloud environment and uses a deep learning approach to justify software prediction. A cost penalty term is added to the supervised part of the deep ladder network; that is, the misclassification cost of different classes is added to the model. A cost-sensitive deep ladder network-based software defect prediction model is proposed, which effectively mitigates the negative impact of the class imbalance problem on defect prediction. To address the problem of lack or insufficiency of historical data from the same project, a flow learning-based geodesic cross-project software defect prediction method is proposed. Drawing on data information from other projects, a migration learning approach was used to embed the source and target datasets into a Gaussian manifold. The kernel encapsulates the incremental changes between the differences and commonalities between the two domains. To this point, the subspace is the space of two distributional approximations formed by the source and target data transformations, with traditional in-project software defect classifiers used to predict labels. It is found that real-time defect prediction is more practical because it has a smaller amount of code to review; only individual changes need to be reviewed rather than entire files or packages while making it easier for developers to assign fixes to defects. More importantly, this paper combines deep belief network techniques with real-time defect prediction at a fine-grained level and TCA techniques to deal with data imbalance and proposes an improved deep belief network approach for real-time defect prediction, while trying to change the machine learning classifier underlying DBN for different experimental studies, and the results not only validate the effectiveness of using TCA techniques to solve the data imbalance problem but also show that the defect prediction model learned by the improved method in this paper has better prediction performance.

1. Introduction

With the rapid development of computer technology, software applications have expanded to all parts of people's daily lives, creating a situation in which the economy, production, and life are fully dependent on computer software. But software failure can bring about serious or even fatal consequences, especially for high-risk systems. System failure is more often caused by software defects, which are important factors affecting software quality and are potential root causes of errors and failures in the relevant systems. In the early stages of software development, due to the limited processing power of computer hardware, software popularity is not high, and the functions realized by the software are relatively simple, so software was mostly developed in an individualized manner [1]. Although the collected data is concentrated, most of the malicious samples are 64-bit programs, but there are still 417 32-bit programs. However, with the development of hardware technology and the popularity of computers, the scale and complexity of software became larger and larger, and the previous way of software development became increasingly difficult, and, to solve the resulting "software crisis," the way of software development was gradually systematized and engineered [2]. However, in the software development process, the existence of software defects is inevitable due to the limitations of resources and developers' experience. Therefore, the prevention of software defects and the timely correction of

software defects have become the reason for the prediction of software defects. People started to collect and use data related to software defects (e.g., data describing the size, complexity, and process of software) and use the data combined with algorithms to construct software defect prediction models for predicting defect information of software.

At the internal level of software, defects are errors or faults in the software development or maintenance process; at the external level of software, defects are violations or failures of the functions that the software needs to perform. Defects in software affect software quality, and early detection of defects and their treatment are important for software quality assurance [3]. However, with the increasing complexity of software systems and the increasing cost of testing, traditional software testing and quality assurance techniques can hardly meet the current needs. If the software development and testing stage can be combined with machine learning technology to deep search, crawl, and analyze the software's historical defect data, to predict and count the distribution and number of defects in the software system in advance to a certain extent, it can better help the quality assurance team to understand the software quality status timely, accurately, and objectively and effectively allocate testing resources to improve software testing efficiency and save testing costs. This can improve software testing efficiency, save testing cost, and guarantee software quality. Through the study of memory analysis technology, a control flow transfer graph generation algorithm based on system calls is proposed. The algorithm is based on the traditional program control flow graph combined with the system call information to automate the generation of system call transfer graph, which can effectively detect and identify malware by using the existing graph neural network model [4]. When it is greater than 0.7, it shows an increasing trend, and the fine-tuned prediction model can show good prediction performance under different sparsity parameters, and the recall rate and F1-measure are better than the performance of the non-fine-tuned prediction model. By mining the software history repository, extracting program modules and metrics, marking whether they contain defects or the number of defects, building software defect prediction models using machine learning and other methods, and then predicting the propensity of defects, defect density, or number of defects for new program modules, software defect prediction technology can be developed [5]. Software defect prediction technology gives software development teams one more chance to retest software defective modules, and, by spending more effort on defect-prone program modules and less on non-defect-prone program modules, the resources of software projects will be better utilized, which can also greatly reduce the human and material resources consumed by testing work, save testing costs, and improve R&D efficiency.

For the core elements involved in software defect prediction techniques, namely, data and algorithms, artificial intelligence plays a great role. When humans first designed programmable computers, they were already thinking about whether computers could become intelligent. Now, artificial intelligence (AI) has many practical applications, is an active research topic, and is flourishing. We expect software to intelligently handle routine labor, understand speech or images, and support basic scientific research. In the early days of AI, problems that were very difficult for human intelligence but simpler for computers were rapidly solved, for example, those that could be described by a set of formal mathematical rules. The challenge for AI is to solve tasks that are easy for a human to perform but difficult to describe formally, such as recognizing what a person says or an image. For these problems, it is often easy for a human to solve them by intuition. Whereas abstract and formalized tasks are among the most difficult mental tasks for humans, they are among the easiest for computers. A key challenge for artificial intelligence is how to communicate nonformal knowledge to computers. Some AI projects seek to hard-code knowledge about the world informal language. Computers can automatically understand this formal language using logical inference rules.

2. Current Status of Research

Currently, the available software defect prediction techniques can be simply classified into dynamic defect prediction techniques and static defect prediction techniques according to the techniques used. Among them, the dynamic defect prediction technique is a study of the entire software system life cycle and predicts the distribution of software defects over time, based on the time when the software failure or system failure occurs, while the static defect prediction technique is based on the size of the software system, loop complexity, and other metric data that have relevance to software defects, as well as the propensity of software program modules to have defects, defect density, or a number of defects prediction [6]. Compared to other machine learning algorithms, the prediction accuracy of SVM is higher, but its algorithm complexity is higher and its operating speed is slower [7]. In addition, since there is no unified theoretical guidance for parameter selection in SVM, many researchers have combined parameter search algorithms with SVM and proposed various software defect prediction models that optimize SVM. Combining the parameter-seeking ability of ant colony optimization algorithm and the nonlinear operation ability of SVM improves the classification performance of SVM; the better global search ability of genetic algorithm is used for the selection of optimal features and the calculation of optimal parameters of SVM, which can avoid the premature sieving of beneficial information in feature selection and further improve the performance of prediction models [8]. It inputs software defect features and historical defect data into the artificial neural network model, compares the error between the output results and the actual results, and corrects the data using a back-propagation algorithm to adjust parameters such as the connection weights of ANNs by an iterative method to continuously optimize and obtain the optimal network parameters [9]. This method has the feature of high prediction accuracy, but the training speed is slow because of the need to iteratively optimize the network parameters. Based on the traditional ANN, the ANN improved first

initializing the parameters of the ANN using a particle swarm optimization algorithm and then using a simulated annealing algorithm to modify the weights and thresholds of the network, which effectively improves the accuracy and precision of the software defect prediction model [10].

Different types of software metrics have been proposed, and many machine learning and data mining algorithms have been applied to solve the problems that arise in the process of software defect prediction based on machine learning algorithms [11]. The cost of classifying defective instances as nondefective instances is too high, thus affecting the usefulness of the prediction model; the high redundancy in the software metric and the high similarity between nondefective instances allow the data quality to be improved by methods such as feature selection [3]. The three methods, as researchers imaginatively call them, analyze software testing in terms of its length, volume, and structure, respectively [12]. Dynamic software defect prediction refers to the technique of predicting the distribution of system defects over time based on the time of defect generation or failure. Most of the researchers study static software defect prediction techniques and, in this paper, also the static software prediction techniques are mainly studied [4]. The study of static software defect prediction techniques is divided into three main aspects: first, how to evaluate software defect prediction models; second, for the problem of choosing software metrics, effectively choosing metrics applicable to software defect prediction; and third, which qualitative or quantitative or hybrid models can be applied to software defect prediction [2, 13]. Regarding evaluation metrics, precision, clarity, and sensitivity are frequently used evaluation metrics.

Precision refers to the proportion of correctly predicted modules to total modules, clarity refers to the proportion of defective modules predicted as defective, and sensitivity refers to the proportion of all modules without defects that are correctly classified. These three metrics do not achieve a comprehensive evaluation of a model, but they enhance the understandability of the model and facilitate further summarization of the model to improve it. Of course, in practical defect prediction, testers can identify some defective modules based on some testing rules. Programmers are also able to identify a set of defect-free modules based on their programming experience. Thus, the initially marked defective modules can be obtained through the efforts of testers and programmers. Using limited defect marking data to predict the propensity for defects in other modules, researchers typically use a limited number of defective modules and many unlabeled modules to be predicted to construct a predictive model using semisupervised learning. Semisupervised learning methods have the advantage of being able to use both labeled and unlabeled data to make use of as much valid data information as possible.

3. Analysis of Deep Learning Software Defect Prediction Methods for Cloud Environments

3.1. Deep Learning Prediction Methods for Cloud Environments. Researchers generally agree that there is a link between internal properties of the software (e.g., static code features) and its external performance (e.g., defects) and that developers collect historical data from the same project or from other projects in their own company and similar projects in other companies, from which they extract static properties of the code (usually expressed using software metrics) and thus have data on the relevant features of the software, and then research can be conducted based on statistical methods. Machine learning is devoted to the study of how experience can be used to improve the performance of the system itself utilizing computation [1]. Therefore, the performance of the prediction model is stable, but, from the running time comparison chart, the running time of the prediction model basically increases linearly with the number of trees in the forest. In computer systems, the experience usually exists in the form of data, and therefore the main part of machine learning research is about algorithms that generate models from data on a computer, that is, learning algorithms. Linear regression can be extended to the classification case by defining different probability distributions; and here the continuous random variables in logistic regression obey the coordination distribution, which means having the following distribution function and density function:

$$p\left(y.x^2;\theta\right) = N\left(y;\theta^2 x,1\right),$$
$$F(x) = \frac{1}{1+\left(-b \pm \sqrt{b^2-3ac}\,/2a\right)}. \quad (1)$$

A feature is an abstract representation of some property extracted from an entity and can consist of data or text; in this paper, features refer to the software defect feature metric already detailed in the previous section. One very important phase that precedes the construction of machine learning models is feature engineering [14]. Feature engineering is the process by which software engineers apply their expertise and skills to analyze and process the dataset to make feature data more useful for machine learning modeling. Subset evaluation is required after the feature subset candidate table is constructed to evaluate the merits of the subset to select the optimal subset. The evaluation methods include the following: the distance measure in FFS, which analyzes the correlation between features and features or features and classes by commonly used measures such as Euclidean distance and squared distance; the information measure in FFS, which evaluates the strength of interdependencies between features and gives a reference for feature selection; the dependency measure in FFS, which gives the separability of features and classes; the consistency metric, which uses inconsistency rate to select the optimal subset; and the classification error rate metric in WFS, which has higher accuracy in evaluating the subset compared to the metric in FFS but takes more time, as shown in Figure 1.

The stopping criterion controls the generation of subsets, which is related to the subset evaluation criterion or search strategy [6]. The hidden neurons of the autoencoder need to be activated too much, which causes the autoencoder training to be overfitted, so the performance of the prediction model is reduced; but when the sparsity parameter is

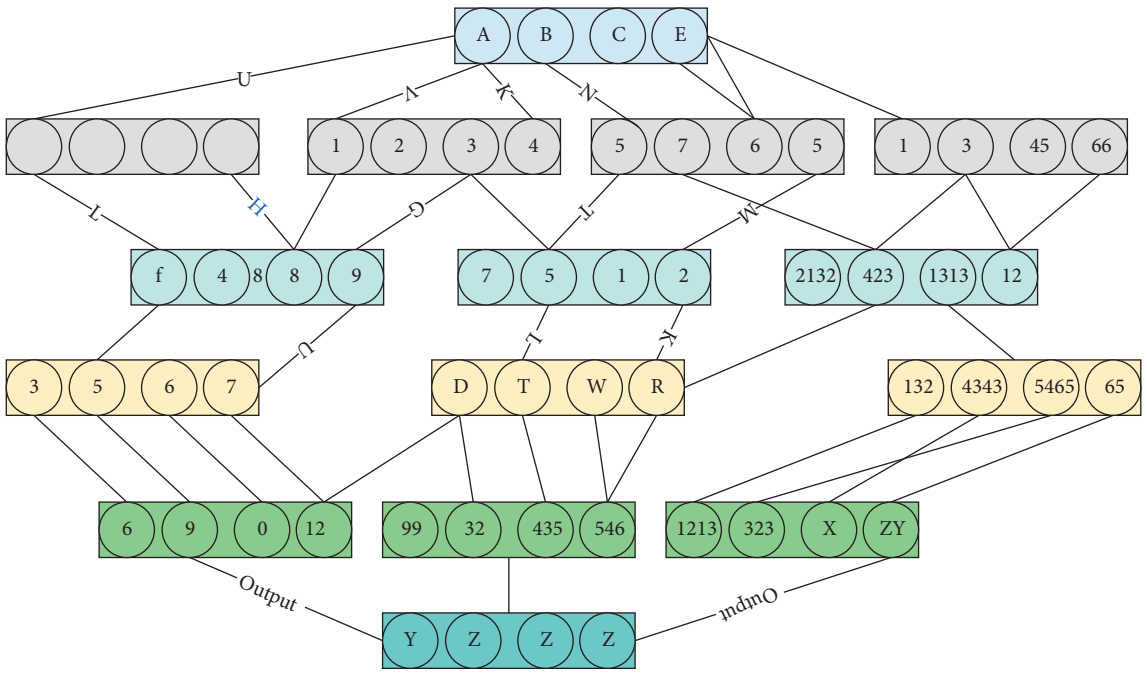

Figure 1: Deep stacking network structure.

greater than 0.7, it means that most neurons in the hidden layer need to be activated this unreasonable setting will destroy the normal optimization of network parameters. The stopping criterion generally includes execution time limit, iteration number limit, and threshold setting. When the global optimal search strategy deals with datasets with many features, its time complexity far exceeds that of other search strategies, and it will greatly affect the efficiency of subset generation, which in turn affects the construction of the whole model while performing a time limit can forcibly end the search, but performing a time limit will affect the optimal subset search performed by this strategy. The stochastic search strategy reduces its time complexity by limiting the number of iterations. The heuristic optimal search strategy prevents the algorithm from entering a dead loop by setting a reasonable threshold.

$$y^3 = f\left(ab_2^3 - \sqrt{a^2 + b^2}\right). \tag{2}$$

Feature extraction is a key step in software defect prediction, and the quality of feature extraction determines the performance of software defect prediction, but traditional feature extraction methods are difficult to uncover the deep-seated essential features in software defect data. Autoencoder models in deep learning theory can automatically learn features from the original data and obtain feature representations of the input data. In this paper, we apply autoencoder to feature extraction for software defect prediction and to improve the loss cost function and sparsity constraint method of autoencoder for its slow convergence of loss cost function and too many sparse regularization parameters, complicated tuning parameters, and so forth. Meanwhile, to reduce the influence of noise on the input data, we propose a software defect prediction based on stacked noise-reducing sparse autoencoder method, which can automatically learn features from the original defect data, extract the required features at each level from the software defect data directly and efficiently by setting different hidden layers, sparse regularization parameters, and noise addition ratios, and then combine with a logistic regression classifier to classify and predict the extracted features.

$$\begin{aligned} \theta^* &= \arg\max \sum_{j+2} nL\left(x_j^{(i)}, y_m^{(i)}\right), \\ J_{SE}(x, y) &= \frac{1}{n} \sum_{y-1}^{n} N\left(\frac{1}{2}\left(\|m\|^3 + \|m'\|_2^3\right)\right). \end{aligned} \tag{3}$$

The inclusion or not of system call execution is added as code block node information in the control flow graph. Some of these nodes do not execute system calls, and this paper is concerned with system call execution information that reflects the intent of the program behavior; such a node is redundant for the graph and needs to be fused with neighboring nodes that contain the system calls. The complexity of the graph is further reduced by generating a control flow transfer graph based on system calls. It propagates labels from labeled data to unlabeled data [15]. The basic idea is that, given a finite amount of labeled data, the labels of the labeled data are propagated through a dense region of unlabeled data, searching for more data with similar properties to the labeled data. A straightforward approach is to compute pairwise similarities between data points and then transform the problem into a harmonic energy minimization problem.

$$\begin{aligned} w_{ab} &= e\left(\left\|a_i - b_y\right\|^3\right) / -(5\pi), \\ &\max \left\|Xm^2 - y\right\| + \pi m^3 \le 3. \end{aligned} \tag{4}$$

The basic assumption of graph-based semisupervised learning is the cluster assumption. It states that if there is a path that connects two points only through a high-density region, then the two points may have the same class label. In computing nonnegative sparse weights for all samples, a sparse matrix can be constructed. With traditional machine learning classification models, the model is often built based on a condition that the data classification is balanced; however, when the data class distribution is unbalanced, it can affect the classification effect. The Pd value increased by 0.07, 0.07, 0.06, and 0.07, respectively. SOM-ANN uses autoencoders to extract data features and trains neural network classifiers to predict defects in software modules. Classifiers that classify classes based on decision surfaces in the feature space can be affected by unbalanced data class distribution. To reduce the effect of noisy data as well as overfitting, the optimal decision surface considers both the accuracy of the classification and the complexity of the decision surface, also called the structural risk minimization principle. However, if the training sample dataset class distribution is unbalanced, in this case, the number of support vectors is also unbalanced. With the principle of structural risk minimization, the support vector will ignore the effect of rare classes on the structural risk and expand the bounds of the decision. Finally, it will lead to a large gap between the actual hyperplane of training and the optimal hyperplane, as shown in Figure 2.

Based on the actual defect prediction needs, defect prediction models are constructed based on the training set and selected machine learning methods (e.g., plain Bayes, support vector machines, linear regression, etc.); the constructed models can be used to perform defect prediction for non-training-set instances (e.g., unlabeled instances or new instances) in this project; the goal of prediction is to discover whether the instances have defects or the number of defects, and this paper studies the former; that is, the instances with or without defects are predicted. To address the problem that the sparsity parameters need to be set in advance and all neurons in the hidden layer have the same sparsity in the traditional sparse autoencoder, a sparse autoencoder with sparsity constraints based on L1 rules is proposed by penalizing the nonzero activation of hidden neurons in the autoencoder, inspired by the sparsity coding algorithm with L1 rules.

Since the different layers of the stacked noise-reducing sparse autoencoder are learned separately, they must be combined with the classifier to form a complete deep neural network model with fine-tuning of the network parameters. Fine-tuning is done by adjusting the parameters of all layers in the model simultaneously by gradient descent to improve its performance after the pretraining process is completed. Fine-tuning is done by removing the decoding layer of the stacked noise-reducing sparse autoencoder, inputting the features of the last layer directly to the classifier for classification, calculating the loss cost of the predicted class versus the actual class, and iteratively optimizing the network parameters by a back-propagation algorithm using gradient descent to obtain a deep model with optimal network parameters.

3.2. Software Defect Prediction Method Design. Increasing the number of trees will not increase the learning performance. Deep belief networks, an example of the deep learning methods, have the greatest advantage over logistic regression in that deep belief networks can generate more expressive sets of features from an initial set of features. If the input is these generated features rather than the initial set of basic features, the two weaknesses that logistic regression has above can be overcome. Based on migration component analysis to deal with imbalanced data, according to the Introduction, the principle of migration component analysis is that even if the distribution of features in the domain is different, when there are some common underlying factors between the source and target domains, the domains can be put into a potential space to explore the hidden common factors in this way to reduce the domain differences, while the original data characteristics can be retained. Therefore, this paper takes advantage of TCA to map the datasets of two different items to a potential space, and when this potential space is found, a few classes of data from the source item are selected and added to the target item as the training set of the target item, so that the two classes of data in the target item are equal in number, and the imbalance of the dataset is improved at this time, and the loss of information from the deleted data is avoided [16]. The imbalance of the dataset is improved, and the situation of losing information by deleting data is avoided. In contrast, when training the target project dataset (intraproject defect prediction), the dataset needs to be mapped to the potential space first, and then the training model is learned in the potential space. Specifically, for the practical application of defect prediction in this paper, the steps are as follows: suppose that the defect prediction model is to be learned in project when the dataset in project 1 is extremely unbalanced; that is, there is less data with defects and more data without defects, as shown in Figure 3.

Although Sigmoid + SVM has the best checking accuracy, its checking completeness rate is very low. This is especially true for the Mozilla, Eclipse JDT, and Eclipse Platform datasets, but the percentage of defective change instances for these three datasets is about 5%, 14%, and 14%, respectively, for which Sigmoid + SVM has only about 4%, 3%, and 3% find-all rates, while Deeper and the improved DBN method achieve a better find-all rate of over 65%. The results show that Sigmoid + SVM is not good enough for defect prediction, illustrating that imbalanced data preprocessing is necessary and important, and Figure 3 demonstrates that addressing the imbalance in the dataset is critical to the prediction performance of the model. When comparing the improved DBNs with the Deeper approach, it is found that the accuracy of almost all the improved DBNs is higher than the accuracy of Deeper. However, only on one dataset, PostgreSQL, the F-measure metric of the Deeper method is larger than that of the improved DBN, which is due to the relatively low accuracy of the improved DBN method in this paper on the PostgreSQL dataset. However, given the defect prediction setting, the check-all rate is relatively more important than the check-accuracy rate; that is, trying to find as many defective changes as possible

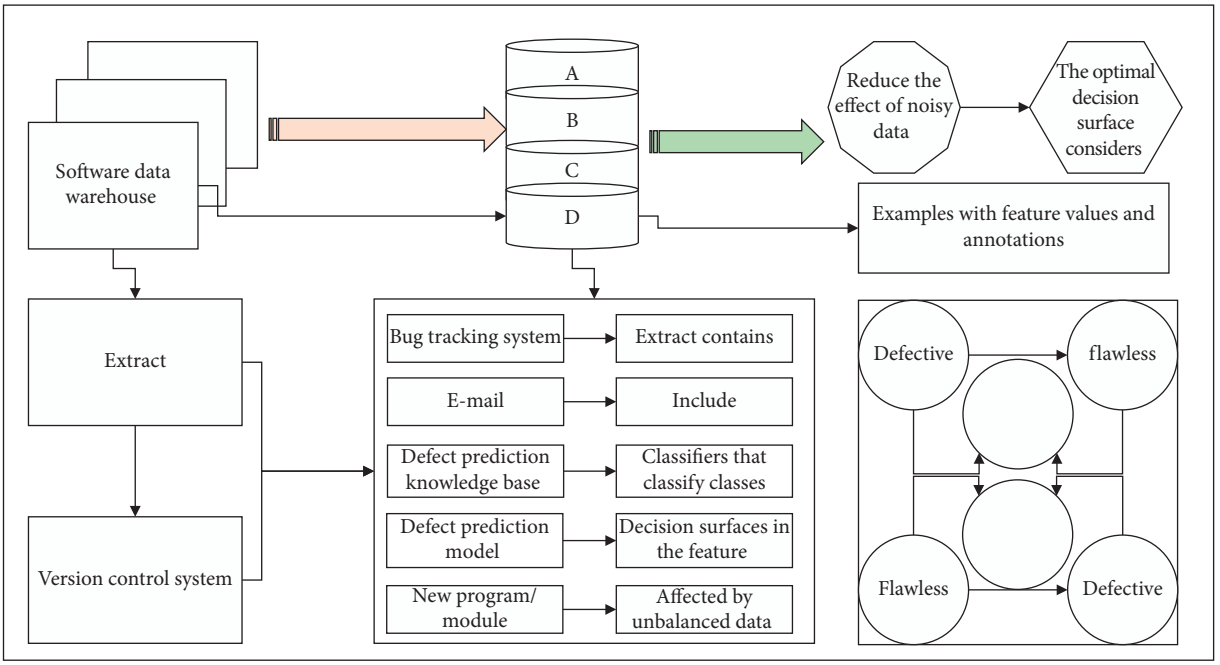

Figure 2: Steps in software defect prediction.

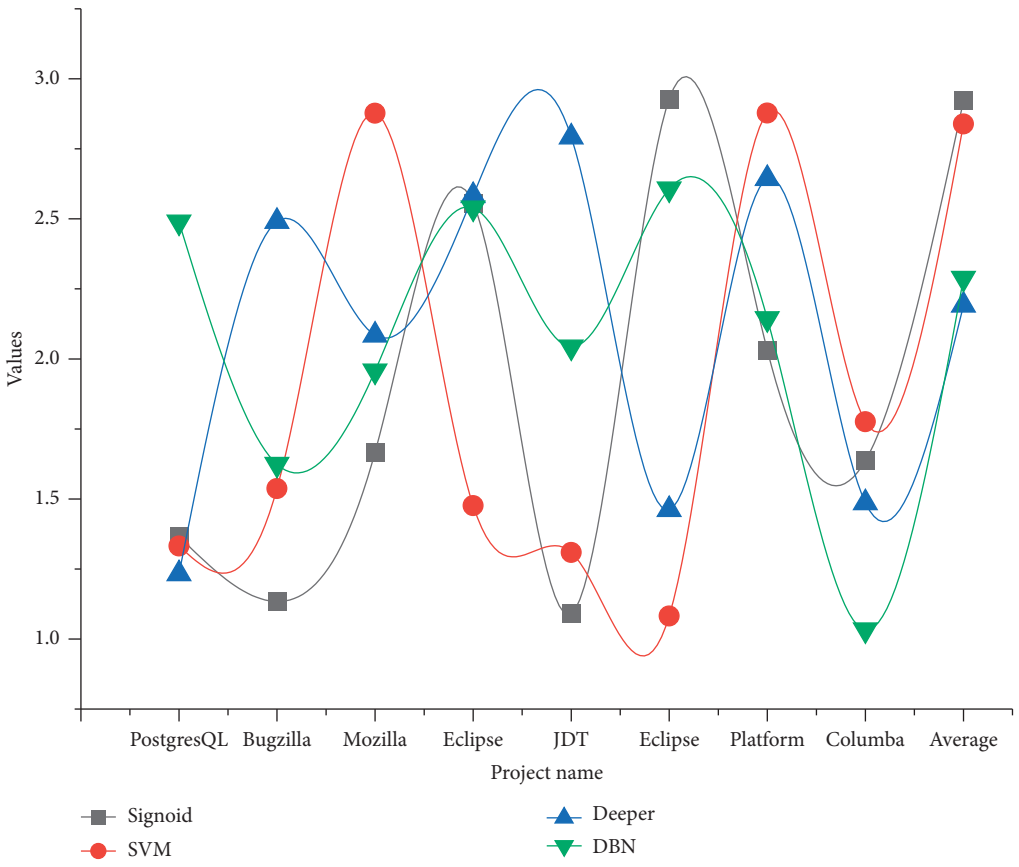

Figure 3: Chakra accuracy.

requires sacrificing some more checking costs to check nondefective instances. Therefore, the improved DBN in this paper is useful and, overall, more effective than the Deeper approach.

The model parallel approach is executed by partitioning the model and putting different computational parts of the model on different machines for the same training samples. For example, the input layer to the first hidden layer of the neural network model is assigned to machine 1, the first hidden layer to the second hidden layer is assigned to machine 2, and the third hidden layer to the output layer is assigned to machine. The disadvantage of model parallelism is that if one of the assigned machines fails, the entire model will stop. The entire dataset consists of a total of 66,301 malware replay images, amounting to 1.3 terabytes of data. The entire dataset was briefly filtered and cleaned before formal analysis. While most malicious samples in the dataset were 64-bit programs, there were 417 32-bit programs in the collection. Although all malware samples run on 64-bit operating systems, all 32-bit images generated by these malicious programs were excluded for data consistency; secondly, the validity of this dataset was checked. By playing back all images and checking the instruction execution during the entire runtime of the malware, it was found that as many as 3462 (5.2%) of the samples did not execute any instruction during the entire runtime. This could be caused by an error during the loading of the malware into memory or by the absence of a necessary component of the target virtual machine operating system of the runtime environment, which prevents the malware from executing correctly. This part of the sample set was also excluded, as shown in Figure 4.

In terms of prediction accuracy, recall, and F1-measure, the prediction model without fine-tuning shows a monotonic increasing trend when the sparsity parameter is less than 0.4, a decreasing trend in the interval of 0.4–0.7, and an increasing trend when it is greater than 0.7. The fine-tuned prediction models show good prediction performance with different sparsity parameters, and the recall and F1-measure are better than the performance of the untrimmed prediction models. To analyze the reason, the number of hidden neuron activations of the autoencoder increases with the increase of the sparsity parameter when the sparsity parameter is less than 0.4, so that the autoencoder can learn the original data with more detailed feature representation and get more essential feature representation of the original data; therefore, the performance of the prediction model increases with the increase of the sparsity parameter. When the sparsity parameter is between 0.4 and 0.7, too many hidden neurons of the autoencoder need to be activated, resulting in overfitting of the autoencoder training, so the performance of the prediction model decreases; but when the sparsity parameter is greater than 0.7, it means that most of the neurons in the hidden layer need to be activated, and this unreasonable setting will destroy the normal optimization of the network parameters, although the prediction model can still achieve the prediction ability. Therefore, prevention of software defects and timely correction of software defects have become the cause of software defect prediction. People began to collect and use data related to software defects and use data combined with algorithms to construct software defect prediction models for predicting software defect information. Although the prediction model can still achieve the prediction ability, the parameters are not obtained through "learning," so the prediction model is an invalid prediction model, and the prediction result is not a valid prediction result. In contrast, the network parameters of the fine-tuned prediction model have been adjusted under the supervision of the classifier, so that the effect of the sparsity parameter setting of the autoencoder on the performance of the prediction model disappears; therefore, the fine-tuned prediction model can show good performance under different sparsity parameters.

4. Analysis of Results

4.1. Deep Learning Predictive Analytics for Cloud Environments. From Figure 5, the nonnegative sparse graph-based label propagation prediction (NSGLP) method has significantly improved the prediction results compared to SVM, CC4.5, and NB; however, compared to the method proposed in this paper, ILR-SDP method, the Pd values of this paper's method based on NSGLP are improved on CM1, MW1, PCL, PC3, and PC4 by 0.09 and 0.08. Because the NSGLP method uses the label propagation method to predict labels on unlabeled data, the predicted obtained labels are used in the trained prediction model, and predicting wrong labels will affect the accuracy of the model. In contrast to the NSGLP method, the ILR-SDP method in this paper uses both real and valid labeled and unlabeled data to train the trapezoidal network. The method in this paper improves the Pd values by 0.07, 0.07, 0.06, and 0.07, respectively, compared to the SOM-SDP method. SOM-ANN uses self-encoders to extract data features and train neural network classifiers for defect prediction of software modules. Compared to these methods, this method uses noise-reducing self-encoders and adds lateral jumps to them. In addition, a small amount of labeled data is used for training. The experimental results prove that the method in this paper does outperform the SOM-ANN method which only uses the traditional self-encoder.

It follows that, in most combinations, JDM can improve the performance of cross-project software defect prediction through a single iteration, and the pseudolabeling refinement mentioned in the method can further improve or consolidate the cross-project software defect prediction performance of the model; the number of convergences of JDM varies from combination to combination due to the differences in both the instance and labeling distributions across combinations; furthermore, not all combinations can be improved by the JDM. One possible reason for this is that the conditional probability distributions used in JDM are computed from pseudoannotations rather than real annotations, and thus the computation of conditional probability distributions may be biased, leading to some degree of degradation in prediction performance. In practical use cases, the efficiency of the detection method and the performance load introduced on the target VM also need to be

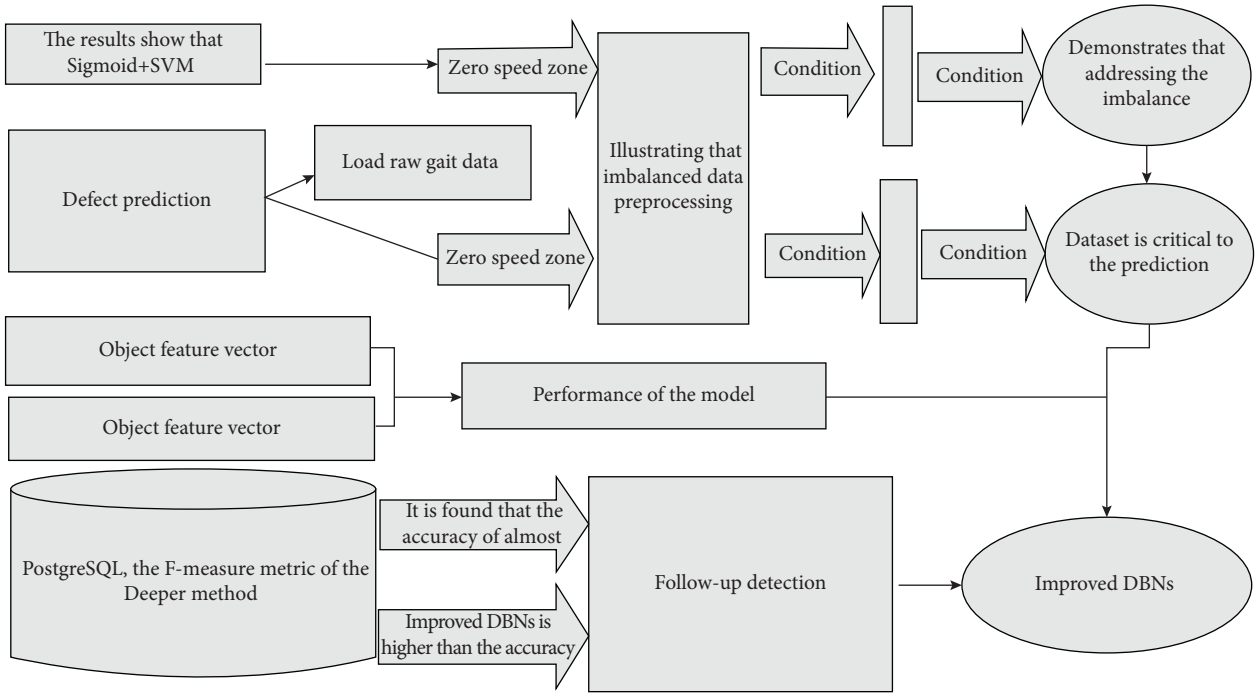

Figure 4: Model parallel framework.

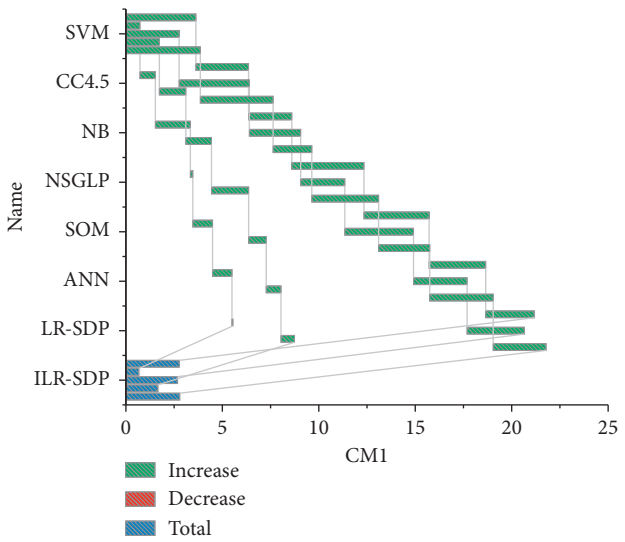

Figure 5: Experimental results of the ILR-SDP method in the NASA dataset.

measured. Improve software testing efficiency, reduce testing expenses, and ensure software quality by assisting the quality assurance team in understanding the software quality status in a fast, accurate, and objective manner and effectively allocating testing resources. If the additional load is high, it may even lead to the unavailability of the target VM. Finally, this paper tests the performance load of each method on the target VM. For the prediction model with the same masking rate, the more hidden layers it has, the higher the correct prediction rate is, and the better the prediction performance in terms of prediction accuracy, recall, and F1-measure is; meanwhile, for the prediction model with the same hidden layers, the prediction performance remains the same when the masking rate is less than 30%, and when the masking rate is greater than 30%, as the masking rate increases, the performance of the prediction model shows a decreasing trend, as shown in Figure 6. Analyzing the reason, for the same masking rate of the prediction model, the deeper the model can obtain the input data deeper feature information, the deeper the layer to obtain the feature information has a stronger feature expression ability, so the more hidden layers of the prediction model prediction performance are better. However, the number of hidden layers is not better but depends on the scale of the prediction model and the training data.

When the prediction model is too deep, it will cause the phenomenon of "gradient disappearance," so that the model training cannot be completed and the prediction results will not be obtained. Therefore, the number of hidden layers should not be increased blindly. For the prediction model with the same number of hidden layers, when the masking rate is less than 30%, the masking noise will remove the noisy data in the training data, which can make the learned prediction model have more robust generalization ability; when the masking rate is greater than 30%, the masking noise will remove the data in the training data that are relevant to software defects, that is, the information that is useful for defect prediction, so the larger the masking rate, the worse the performance of the prediction model. A software defect prediction model is built with the help of machine learning and other methods, and then the defect tendency, defect density, or number of defects of the new program module is predicted. Software defect prediction technology gives the software development team one more opportunity to redetect software defect modules.

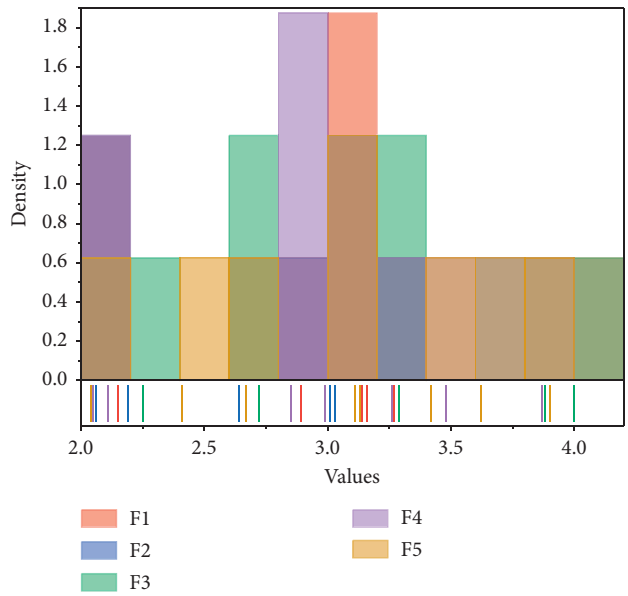

Figure 6: Algorithm performance.

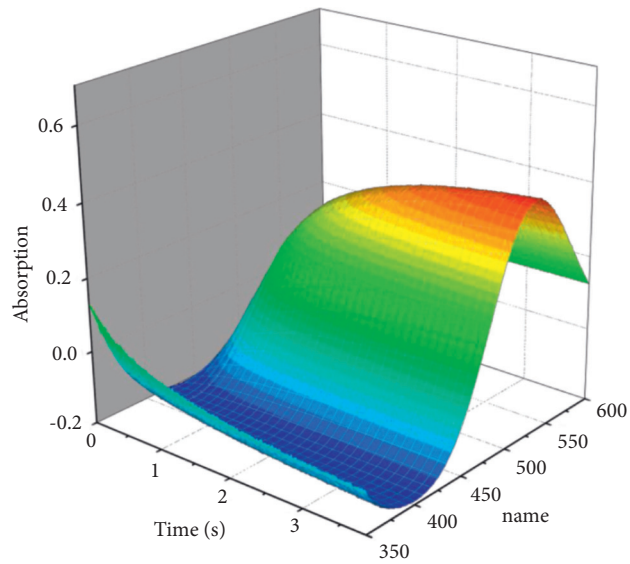

Figure 7: Comparison of F1-measure results for NASA dataset.

4.2. Software Defect Prediction Results. According to the summary of the experimental results, it can be obtained that the experimental metric F1-measure of TNB and the proposed algorithm GFK-SDP in this paper is higher than NN-filter. This is caused by the disadvantage of the NN-filter algorithm in the paper, in which the training dataset is selected during the construction of the algorithm, and this selection process may cause the loss of useful data information, so the final F1-measure value is lower than those in the other two methods. Besides, the GFK-SDP method is significantly better than the TNB method, thanks to the introduction of the geodesic kernel space transformation technique. In the new subspace representation, the distributions of the source and target data are approximated. This effectively reduces the error caused by the inaccurate model training effect due to different data distribution of different datasets and at the same time effectively maximizes the useful feature information in the source and target data. On this basis, the results were effectively improved using the traditional classical classification method, as shown in Figure 7.

Learning-based cross-project software defect prediction techniques have now become a technique of interest to researchers due to the superior performance of stream learning techniques on cross-project software defect prediction problems. For cross-project software defect prediction techniques, it is an important research topic to solve the problem of a different distribution of data features between source and target projects and to effectively mine the shared information between different projects.

People often find it easy to solve these problems by intuition. Abstract and formal tasks are one of the most difficult mental tasks for humans, but they are the easiest for computers. In this chapter, a cross-project software defect prediction method based on geodesic streamlines is proposed, which draws on the knowledge of incremental learning to effectively solve the problem of different data distribution between different projects. The software defect prediction model is based on the cross-project model, and the source dataset is used as training data to obtain the software defect prediction model, and, finally, the obtained training model is used to test the data to be predicted. The main problem is that the data distribution of the source data and the target data are different, as shown in Figure 8.

As can be seen from the figure, for a stacked forest of the same depth, when the number of trees in the forest is less than 200, the prediction correct rate, accuracy rate, and AUC value all show an increasing trend, and the prediction recall rate shows a decreasing trend, while the prediction F1-measure shows a stable performance. To analyze the reason, when the number of trees in the forest is less than 200, the learning ability of the random forest and completely random tree forest in the stacked forest increases with the increase of the number of trees, so that the stacked forest can learn more detailed data information, and, therefore, the performance of the prediction model increases with the increase of the number of trees in the forest. For these three datasets, the recall rates of Sigmoid + SVM are only about 4%, 3%, and 3%, while Deeper and the improved DBN method achieve a better recall rate of more than 65%. When the number of trees in the forest is greater than 200, the learning ability of the random forest and the completely random tree forest in the stacked forest has reached saturation, and increasing the number of trees again will not increase the learning performance, so the performance of the prediction model is smooth, but it can be seen from the running time comparison graph that the running time of the prediction model basically grows linearly with the number of trees in the forest, so the performance of the stacked forest is the best when the number of trees in the forest is 200. The loss cost function and sparsity constraint method of the autoencoder are improved. At the same time, to reduce the influence of noise on the input data, a software defect prediction method based on the stacked noise-reducing sparse autoencoder is proposed; and, for the same number of

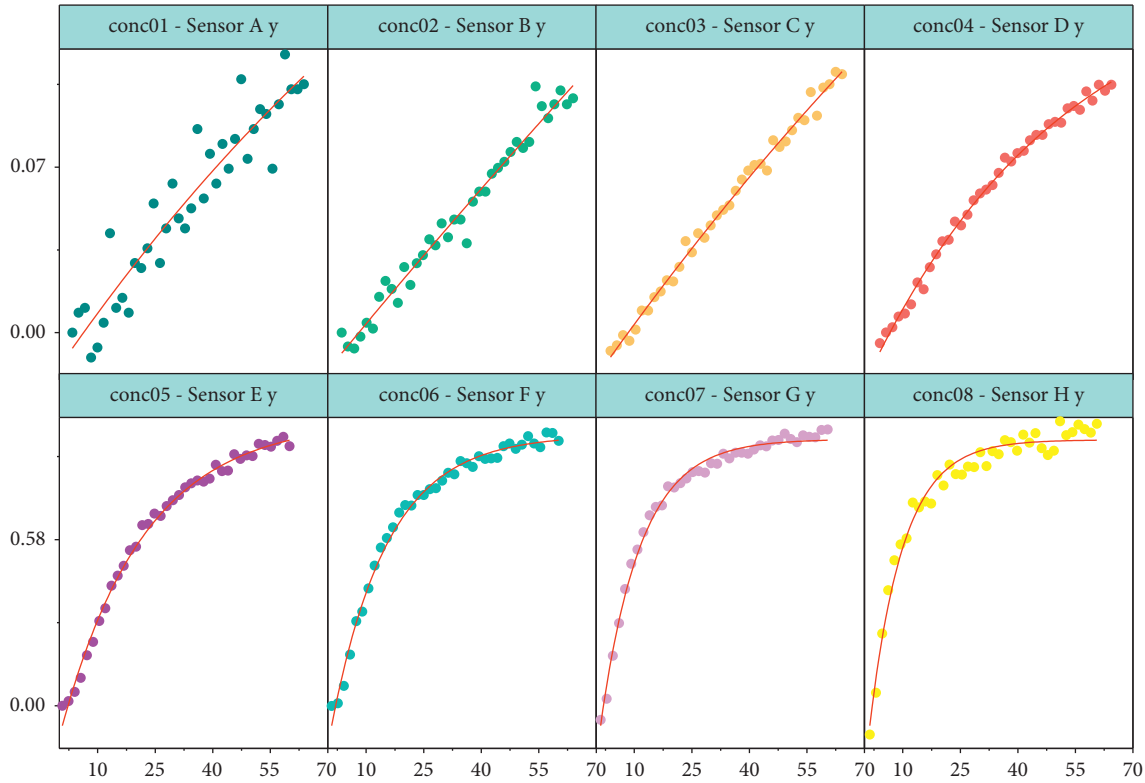

Figure 8: Effect of depth of stacked forest and number of trees on prediction model performance.

trees in the stacked forest, when the stacked forest is deeper, the correct prediction rate remains the same, and the prediction accuracy tends to decrease, but the prediction recall, F1-measure, and AUC values all show an increasing trend, and it can be seen from the comparison graph of prediction AUC values that the performance of the prediction model increases slowly and negligibly when the depth of the stacked forest is greater than three layers so that the stacked forest depth of three layers is the best choice.

5. Conclusion

Deep neural networks require large-scale training data during training, and the task for small-scale data will fail to achieve the desired effect due to training underfitting, and the hyperparameters of deep neural networks are too many, which makes tuning the parameters complicated. To address these problems, the deep forest is applied to software defect prediction, and its feature vector generation method, feature transformation method, and feature enhancement method are improved to address the shortcomings in software defect prediction performance, and the software defect prediction release method based on the deep stacked forest is proposed. The experimental results show that the prediction method based on the deep stacked forest has advantages over the prediction method based on deep forest in terms of prediction performance and operational efficiency. The method of three rounds of manual testing and software defect prediction is used; that is, some program modules are first randomly selected for manual testing, and then the tested program modules are used as training sets to construct software defect prediction models, predict the defect propensity of untested program modules, and then manually test the program modules with defect propensity, and so on. If the class distribution of the training sample dataset is unbalanced, the number of support vectors is also unbalanced. Based on the principle of minimizing structural risks, support vector opportunities ignore the impact of rare classes on structural risks and expand the boundaries of decision-making. After three rounds of manual testing and software defect prediction, the system contains the defect, and the number of defective program modules in the system is reduced to an acceptable level. The combination of manual testing and software defect prediction greatly reduces the number of manually tested program modules, reduces software testing time, and improves software development efficiency. It is practical to apply deep learning methods to malware detection. This paper combines the existing virtual machine introspection technology to migrate all security work outside the virtual machine, which is more secure and efficient; and the use of deep learning methods can also effectively solve the drawbacks of traditional detection methods to make effective detection and early warning of unknown malware variants; differing from single detection methods, this paper effectively combines dynamic/static detection methods.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

All the authors do not have any possible conflicts of interest.

References

[1] X. Zhang, X. Chen, J. Wang, Z. Zhan, and J. Li, "Verifiable privacy-preserving single-layer perceptron training scheme in cloud computing," *Soft Computing*, vol. 22, no. 23, pp. 7719–7732, 2018.

[2] R. M. Alguliyev, R. M. Aliguliyev, and F. J. Abdullayeva, "Hybridisation of classifiers for anomaly detection in big data," *International Journal of Big Data Intelligence*, vol. 6, no. 1, pp. 11–19, 2019.

[3] Y. Zhang, W. Huang, T. Zhang, and T. Zhang, "A novel topology optimization theory and parallel data analysis model based resource scheduling algorithm for cloud computing," *Recent Advances in Electrical and Electronic Engineering*, vol. 11, no. 4, pp. 449–456, 2018.

[4] S. Yang, "IoT stream processing and analytics in the fog," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 21–27, 2017.

[5] L. Zhao, W. Alhoshan, A. Ferrari et al., "Natural language processing for requirements engineering," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–41, 2021.

[6] S. G. Tzafestas, "The Internet of Things: a conceptual guided tour," *European Journal of Advances in Engineering and Technology*, vol. 5, no. 10, pp. 745–767, 2018.

[7] Y. Qi, C. Fang, H. Liu et al., "A survey of cloud network fault diagnostic systems and tools," *Frontiers of Information Technology and Electronic Engineering*, vol. 22, no. 8, pp. 1031–1045, 2021.

[8] Q. Chen, C. Lan, L. Zhao, J. Wang, B. Chen, and Y.-P. P. Chen, "Recent advances in sequence assembly: principles and applications," *Briefings in Functional Genomics*, vol. 16, no. 6, pp. 361–378, 2017.

[9] A. Shaout and C. Smyth, "Fuzzy zero day exploits detector system," *International Journal of Advanced Computer Research*, vol. 7, no. 31, pp. 154–163, 2017.

[10] U. Ali, A. Mehmood, M. F. Majeed et al., "Innovative citizen's services through public cloud in Pakistan: user's privacy concerns and impacts on adoption," *Mobile Networks and Applications*, vol. 24, no. 1, pp. 47–68, 2019.

[11] H. Ma, H. Ding, Y. Yang, Z. Mi, J. Y. Yang, and Z. Xiong, "Bayes-based ARP attack detection algorithm for cloud centers," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 17–28, 2016.

[12] G. Zhang, K. Zhang, X. Zhu, M. Chen, C. Xu, and Y. Shao, "Modeling and analyzing method for CPS software architecture energy consumption," *Journal of Software*, vol. 8, no. 11, pp. 2974–2981, 2013.

[13] J. Zhu, Q. Li, and Y. Shi, "Failure analysis of static analysis software module based on big data tendency prediction," *Complexity*, vol. 2021, Article ID 6660830, 12 pages, 2021.

[14] O. Olugboyega, E. D. Omopariola, and O. J. Ilori, "Model for creating cloud-BIM environment in aec firms: a grounded theory approach," *American Journal of Civil Engineering and Architecture*, vol. 7, no. 3, pp. 146–151, 2019.

[15] A. Kaur, N. Singh, and D. G. Singh, "An overview of cloud testing as a service," *International Journal of Computers and Technology*, vol. 2, no. 2, pp. 18–23, 2012.

[16] T. Naresh, A. J. Lakshmi, and V. K. Reddy, "Resource allocation methods in cloud computing: survey," *International Journal of Engineering Trends and Technology*, vol. 2, no. 2, pp. 416–419, 2015.