Defect Prediction of Finite State Machine Models Based on Transfer Learning

Wei Zhang
Experimental Teaching Center, Shandong University of Finance and Economics, Jinan 250014, China

***Abstract*—As software systems become increasingly intricate, predicting cache defects has emerged as a crucial aspect of maintaining software quality. This article introduces a novel approach for predicting cache defects, utilizing a transfer learning (TL) software deterministic finite state machine (DFSM) model. Finite State Machine (DFSM) model defect prediction based on transfer learning is an innovative software defect prediction method. This method combines the advantages of transfer learning (TL) and deterministic finite state machine (DFSM). Intended to improve the effectiveness and accuracy of software cache defect prediction. This innovative method seeks to enhance the effectiveness of predicting cache issues within software. By merging the precision of DFSM with TL's versatility, the proposed technique is transferable to target projects through training and learning from source projects, addressing data scarcity challenges in new or evolving projects. This method utilizes transfer learning (TL) strategy to transfer knowledge from the source project to the target project through learning and training, thereby solving the problem of data scarcity. Experimental findings reveal that as training data grows, the method's test coverage and fault detection rate steadily increase. Additionally, it demonstrates impressive execution efficiency and stability. In comparison to traditional methods, this approach exhibits substantial benefits in elevating software quality and reliability, offering a fresh and efficient tool for ensuring software quality. Thanks to the TL strategy, the method rapidly adapts to the unique environments and requirements of new or evolving projects, thereby enhancing forecasting accuracy and efficiency.**

Keywords*—Transfer learning; DFSM; software defects; defect prediction

I. Introduction

In the intricate ecosystem of software development, maintaining software quality and stability remains a pivotal concern. As software systems grow increasingly vast and complex, the effective prediction and prevention of software flaws have emerged as a significant hurdle in the realm of software engineering [1]. Software flaws can detract from the user experience, potentially causing substantial financial losses and even posing security risks. Despite ongoing practices in software development, predicting such flaws still poses numerous obstacles. Conventional methods for predicting software flaws often rely heavily on extensive historical datasets, limiting their adaptability to novel projects or environments [2]. Furthermore, these approaches frequently overlook the interconnectedness and disparities among software projects, potentially compromising the accuracy of predictions. However, through the lens of transfer learning (TL), we can harness existing knowledge and expertise, bridging the gap between projects and enabling more efficient and precise flaw predictions [3]. TL facilitates the transfer of insights gained from one task to others within the same domain, thereby enhancing learning efficiency and forecast accuracy [4]. In the realm of software flaw prediction, TL holds tremendous promise. By leveraging existing software project data, TL can aid in predicting flaws in new projects, expediting the model's training process and bolstering predictive accuracy [5].

Under the framework of DFSM, software system can be regarded as a process of state transition. By analyzing and modeling the state transition behavior of software system, we can understand its internal logic and operating mechanism more deeply [6]. In software defect prediction, with the help of DFSM model, the state transition paths that may lead to defects can be identified, thus improving the accuracy of prediction [7]. The traditional DFSM model often needs a lot of historical data in the construction process, and its adaptability to new projects is poor [8]. This article aims to study the cache defect prediction model and algorithm of software DFSM model based on TL. By introducing TL strategy, the existing knowledge and experience can be used to assist the DFSM model construction of new projects, thus improving the generalization ability of the model.

The research of this article has important theoretical and practical significance. By combining TL and DFSM models, we can understand the behavior pattern of software system more deeply, and then predict the potential defects more accurately. The research results can provide valuable reference information for software developers and help them identify and prevent software defects more effectively in the actual development process. The research can also provide new ideas for the field of software quality management and promote the sustainable development of the software industry.

The structure of this article is as follows: Firstly, the current situation of software defect prediction is sorted out in the literature review in Section II, and the advantages and disadvantages of existing methods are analyzed. Then, the basic principle of TL and its application prospect in software defect prediction are introduced in Section III. Then, the construction process and experimental design of cache defect prediction model of software DFSM model based on TL are emphasized. Finally, the experimental results are deeply analyzed in Section IV and a conclusion is drawn in Section V.

The research motivation of this article is to explore a more efficient and accurate software defect prediction method,

especially for predicting cache defects. By combining the advantages of transfer learning and DFSM models, we hope to gain a deeper understanding of the behavioral patterns of software systems. Identify potential defect state transition paths and provide valuable reference information for software developers. The potential benefits of this method are mainly reflected in the following aspects:

1) Through transfer learning strategies, we can utilize existing knowledge and experience to assist in the construction of DFSM models for new projects. Thus improving the generalization ability and prediction accuracy of the model.

2) By predicting potential defect state transition paths, developers can detect and fix defects at an early stage, thereby avoiding the high cost of later repairs.

3) This study not only provides new ideas and methods for software defect prediction, but also valuable references for software quality management.

II. Literature Review

Software defect prediction, a pivotal aspect of enhancing software quality, remains a focal point of research in software engineering. Numerous scholars have delved into this domain from diverse perspectives, presenting a range of methodologies and models.

Conventionally, researchers have heavily relied on historical datasets and statistical analyses for software defect prediction. Zhang et al. [9] introduced a model that forecasts future defect patterns by analyzing past defect records. Gong et al. [10], employing statistical techniques, assessed software project quality, revealing notable correlations between factors like project size, complexity, and the occurrence of software defects. Addressing the specifics of distributed software systems, Wang et al. [11] suggested a cloud-based framework tailored for handling extensive software defect data and delivering swift prediction services. Li et al. [12] innovated a method rooted in differential privacy, balancing effective defect prediction with robust data privacy measures. Wang et al. [13] advanced a decision tree-based prediction model, noted for its high predictive accuracy and intuitive explanations.

In recent times, the utilization of machine learning in predicting software defects has been on the rise. Chakraborty et al. [14] employed the Support Vector Machine (SVM) for software defect prediction, yielding impressive results. Wang et al. [15] conducted a comparative analysis of various machine learning techniques in this domain, discovering that algorithms like Random Forest and Gradient Boosting Tree performed admirably on certain datasets. Yu et al. [16] explored the impact of the social network structure within software projects on defect prediction, revealing a notable correlation between project member collaborations and the occurrence of software defects. Florence et al. [17] introduced a deep learning-based model for software defect prediction, capable of automatically extracting features from software codes and predicting defects. Tong et al. [18] suggested a method rooted in oversampling to balance datasets by augmenting the number of defective modules. Song et al. [19] tackled unbalanced data using ensemble learning, enhancing the recognition of minority classes by amalgamating predictions from multiple base classifiers.

TL as an emerging machine learning technique, has also gained traction in software defect prediction. Saifan et al. [20] proposed a TL-based model that leverages knowledge from source projects to aid in defect prediction for target projects. Qu et al. [21] delved deeper into the application of TL in cross-project software defect prediction, affirming its efficacy. Bashir et al. [22] presented a TL-driven method that allows for real-time model updates during software system integrations, adapting to system changes.

In the realm of DFSM modeling, El-Fakih et al. [23] introduced a method for modeling software behavior based on DFSM, precisely capturing the state transition processes within software systems. Hierons [24] integrated the DFSM model with machine learning algorithms, offering a hybrid approach for software defect prediction. This hybrid method takes into account both the dynamic behavior of the software system and the predictive prowess of machine learning.

At present, the research on software defect prediction faces some problems, such as unbalanced data, poor universality of the model, lack of dynamic adaptability and insufficient explanation. The data imbalance leads to the limited ability of the model to identify minority classes, while the lack of universality of the model makes it perform poorly in new projects or environments. In addition, the continuous evolution of software systems challenges the dynamic adaptability of existing models.

This article aims to build a universal and dynamic software defect prediction model by introducing TL strategy and combining DFSM model to solve the problems of data imbalance and model universality. At the same time, it pays attention to the selection of algorithms with good explanatory ability, and improves the dynamic adaptability of the model through online learning, thus comprehensively optimizing the accuracy and practicability of software defect prediction.

III. TL-based Software DFSM Model Cache Defect Prediction Model

Cache defects are common problems in software development, which may lead to data inconsistency, performance degradation, and even system crashes [25]. In order to more effectively predict such defects, this paper proposes a cache defect prediction model based on TL software DFSM model. This model combines the rigor of DFSM with the flexibility of TL, aiming to achieve rapid defect prediction for new or changed projects. The deployment of the data processing platform for the software testing system is shown in Fig. 1.

A. Foundation of Model Construction

1) DFSM: A DFSM is a mathematical model used to describe system behavior. In software system, DFSM can be used to represent the state transition process of software. Each state represents the specific situation of the software at a certain moment, and the transition between states reflects the behavior changes of the software in the process of running.

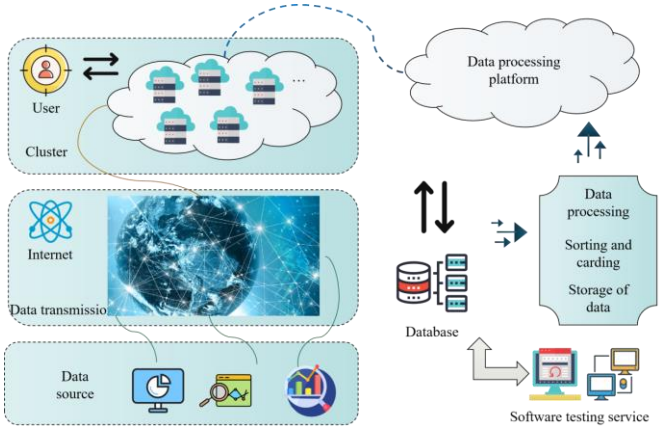

Fig. 1. Deployment of data processing platform of software testing system.

2) TL: TL is a machine learning method, which allows us to transfer what we have learned in one task to other related tasks. In software defect prediction, the application of TL is mainly reflected in two aspects: one is to use the existing software project data to assist the defect prediction of new projects; The second is to transfer the model trained in a software project to other similar projects, to reduce the training cost of new projects and improve the prediction accuracy.

B. Construction of DFSM Model based on TL

1) Data preprocessing and feature extraction: Before building the model, the original data need to be preprocessed and feature extracted. Firstly, the data related to cache is extracted from the source code and log files of software projects. These data include, but are not limited to, the type of cache operation, timestamp, operation result, etc. Then, these data are cleaned and standardized to eliminate the influence of outliers and noise data.

The software distributed parallel computing architecture is shown in Fig. 2.

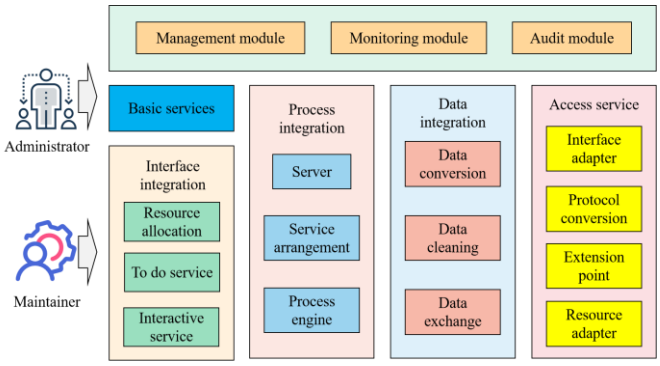

Fig. 2. Distributed parallel computing architecture.

Given that the local data processing capacity is v_i, and considering that the software terminal is capable of processing data locally within the existing resource limitations, it is imperative that the resources allocated for data processing do not surpass its inherent physical resources. In other words, there exist certain constraints:

$$\sum_{i \in N} v_i^{(t)} \le v_{\max}^{(t)} \tag{1}$$

Suppose that the rate at which software test data is migrated to the terminal is:

$$R = W \log_2 \left(1 + \frac{g \cdot P}{N_0 \cdot W} \right) \tag{2}$$

The channel unloading rate R is directly proportional to channel bandwidth W, channel gain g, and transmission power P, while inversely proportional to noise power spectral density N_0. If $a_{ij}^{(t)} = 1$, it signifies that the software testing equipment i is connected to the terminal j, enabling successful data upload to the edge server for processing. Conversely, if the conditions are not met, $a_{ij}^{(t)} = 0$. Additionally, there are specific constraints related to the unloading model of the software test equipment.

$$\sum_{j \in M} a_{ij}^{(t)} \in \{0,1\} \tag{3}$$

$$\sum_{i \in N} a_{ij}^{(t)} \le h_j \tag{4}$$

Eq. (3) represents the maximum number of terminals that a single software testing device can connect to within the same time slot t, which means one software testing device can only be paired with one terminal. Eq. (4) signifies that any given terminal j permits a maximum of h_j software test devices to be simultaneously connected.

In feature extraction, this article mainly focuses on features related to cache defects. These features can include the frequency of cache operations, the proportion of cache hits, and the mode of cache updates. Through in-depth analysis of these characteristics, it is possible to more accurately describe the caching behavior of software systems and provide strong support for subsequent defect prediction.

2) Model migration strategy: In the constructed model, TL is mainly realized by the following steps:

a) Source project selection: First, you need to select one or more source projects similar to the target project. These source projects should have similar cache management mechanisms and defect patterns as the target projects. By selecting similar source projects, we can ensure that the migrated knowledge and experience are instructive to the target projects.

b) Model training and migration: Train a cache defect prediction model based on DFSM on the source project. This model will learn the caching behavior pattern and defect characteristics of the source project. Once the model achieves satisfactory prediction performance on the source project, it can be migrated to the target project.

c) Model adjustment and optimization: During the migration process, some adjustment and optimization are needed to adapt to the specific environment and needs of the target project. For example, fine-tune the parameters of the model according to the data distribution of the target project, or add some features related to the target project to improve the prediction ability of the model.

3) Defect prediction algorithm based on DFSM: After the model migration is completed, the defect prediction algorithm based on DFSM is used to predict the defect of the target project. The algorithm will traverse all the cache operation sequences of the target project, and judge whether there is defect risk in each operation sequence according to the state transition rules of DFSM and the defect characteristics obtained by TL. If there are risks, the algorithm will issue a warning and prompt developers to check and repair. In this section, a model of directly fusing the source syntax tree to the encoder-decoder framework is proposed. The encoder part adopts the cyclic neural network (RNN) of bidirectional gated cyclic unit (GRU), that is, the encoder contains a forward RNN and a reverse RNN (see Fig. 3).

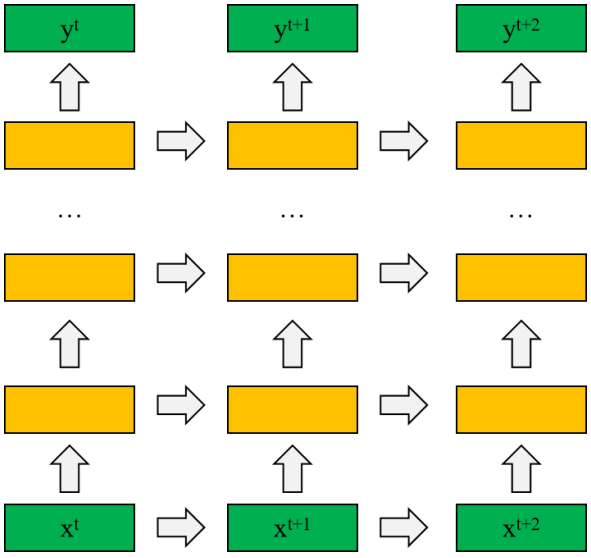

Fig. 3. Example of bidirectional serialization encoder.

Firstly, the original software defect data is preprocessed, including data cleaning, noise and outlier removal, and feature engineering. Define a data preprocessing function P, which transforms the original data set D into a format suitable for model training:

$$D' = P(D) \tag{5}$$

Next, build a TL framework, which allows us to transfer the knowledge learned from other related software projects to the current project. A TL function T is used in the study, which combines the knowledge K_s in the source domain with the knowledge K_t in the target domain:

$$K' = T(K_s, K_t) \tag{6}$$

On the basis of TL, DFSM model is used to represent the behavior of software. Define a DFSM constructor B, which generates the DFSM model M according to the knowledge K' after TL:

$$M = B(K') \tag{7}$$

Based on DFSM model, a defect prediction algorithm A is designed. The algorithm predicts defects according to DFSM model M and current project data D'. Define a prediction function F:

$$P = F(M, D') \tag{8}$$

where, P represents the prediction result, which is a vector containing the defect probability.

In order to train and optimize the defect prediction model, a loss function L is defined. It measures the difference between the predicted result P and the real label Y:

$$L(P,Y) = \sum_{i=1}^{n} (P_i - Y_i)^2 \tag{9}$$

where, n is the number of samples in the data set.

In order to improve the dynamic adaptability of the model, online learning mechanism is introduced. Define an online learning function O, which updates the model parameters according to the newly arrived data D_{new}:

$$M' = O(M, D_{new}) \tag{10}$$

IV. Result Analysis and Discussion

A. Result Analysis

In order to thoroughly evaluate the effectiveness of the software cache defect prediction model based on DFSM and enhanced by transfer learning technology, we designed a series of experiments and selected various software projects of different scales and complexities as experimental objects. Firstly, we collected data related to these software projects, covering software caching behavior, defect records, and other related attributes. By refining and organizing these data, we have constructed a dataset specifically designed for transfer learning.

In the data preparation stage, we ensured the quality and consistency of the data to provide a reliable foundation for model training. Next, we will repeatedly verify according to the pre-defined experimental blueprint. Specifically, we adopted a cross validation approach, dividing the dataset into training, validation, and testing sets. The training set is used to train software DFSM models based on transfer learning, the validation set is used to adjust model parameters and optimize model performance, and the test set is used to evaluate the final performance of the model. The aim of this experiment is to conduct a thorough evaluation of the efficacy of a software cache defect prediction model rooted in DFSM and enhanced by transfer learning technology. The study encompasses a variety of software projects, differing in scale and intricacy, as subjects for examination. Relevant data is gathered, refined,

and organized into a dataset tailored for transfer learning. Through rigorous training and refinement of the model, its performance is gauged across multiple metrics, including test coverage, defect identification rate, algorithmic execution efficiency, and stability. To ensure consistent and dependable outcomes, the entire experimental procedure is executed on a server equipped with ample computational resources.

During the experimental phase, the predefined experiment al blueprint is repeatedly validated. Initially, a prediction model is devised employing the transfer learning-based software DFSM model for cache defect prediction. Crucial data points and assessment criteria are documented throughout. Subsequently, the model undergoes training and optimization, culminating in an enhanced defect prediction model. A detailed breakdown of the experimental findings follows.

As illustrated in Fig. 4, the increase in training data gradually improves the test coverage of the software DFSM model's cache defect prediction method, which relies on TL. This progress indicates that the method is capable of efficiently learning and predicting software cache defects. In comparison to traditional approaches, our method demonstrates notable superiority in test coverage, thereby enhancing the overall quality and dependability of software.

According to the data shown in Fig. 5, the TL-based software DFSM model cache defect prediction method has a high fault detection rate. This means that this method can accurately identify the cache fault in software. Compared with other technologies, this method also has significant advantages in fault detection rate, which is helpful to improve the overall performance of the software and user satisfaction.

Fig. 6 shows the execution time of the algorithm. Although the execution time of the algorithm will increase with the increase of data volume, on the whole, the software DFSM model cache defect prediction method based on TL performs well in execution efficiency. This shows that this method can not only ensure the prediction accuracy, but also maintain efficient operation performance.

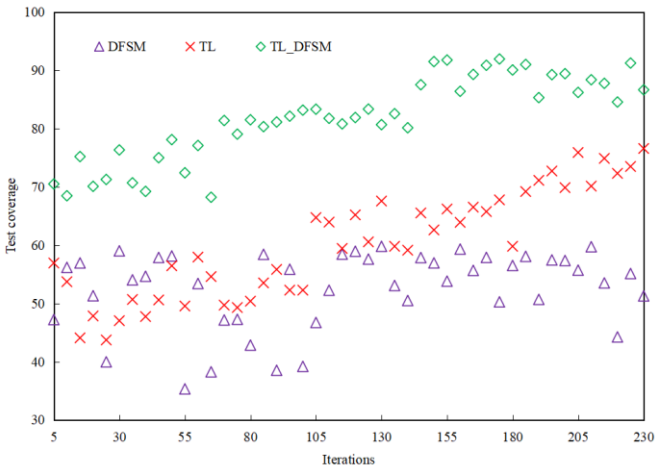

Fig. 4. Test coverage.

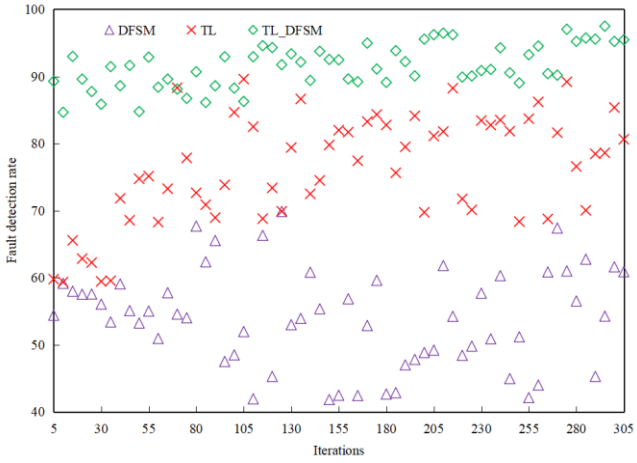

Fig. 5. Fault detection rate.

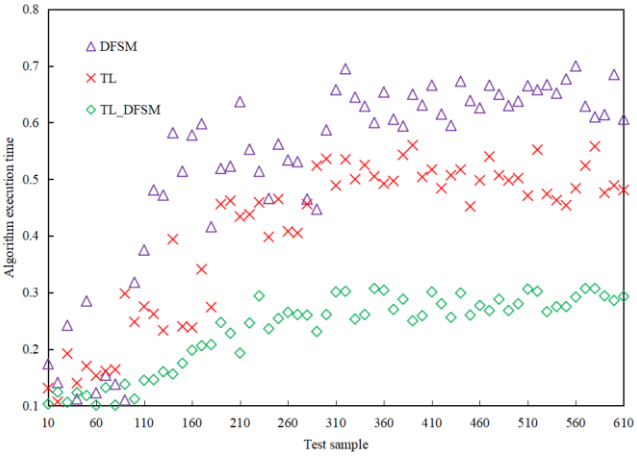

Fig. 6. Algorithm execution time.

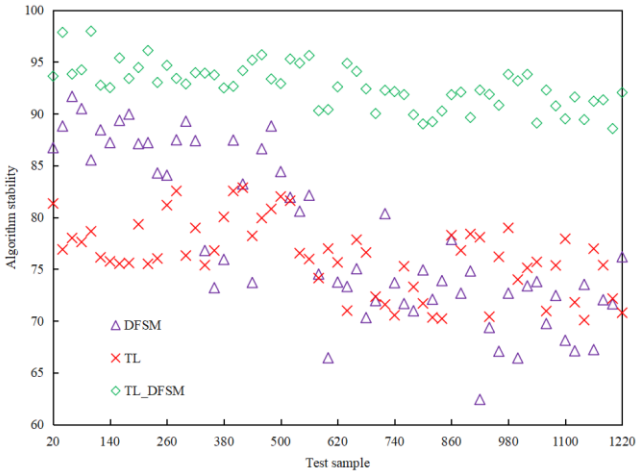

Fig. 7. Stability of algorithm.

As can be seen from Fig. 7, the TL-based software DFSM model cache defect prediction method has excellent stability. Under different experimental conditions, the prediction performance of this method remains stable, and the fluctuation of test coverage and fault detection rate is small. This proves that this method has good robustness and reliability, and is very suitable for practical software defect prediction scenarios.

Through a series of experiments, it is proved that the TL-based software DFSM model cache defect prediction method is excellent in improving test coverage, fault detection rate and maintaining algorithm execution efficiency and stability, which provides a new and effective means for software quality assurance.

B. Discussion

In today's software development field, cache defect prediction is an important and challenging problem. With the increasing complexity and scale of software system, how to effectively predict and identify potential cache defects in order to repair them in time and improve software quality has become the focus of researchers and practitioners. The cache defect prediction model of software DFSM model based on TL proposed in this article provides a new idea for solving this problem.

DFSM provides a clear representation of software systems' dynamic behaviors, including essential operations like cache hits, misses, and updates, by precisely defining states and the transitions between them. This lays a strong theoretical foundation for defect prediction. Additionally, DFSM's interpretability aids developers in intuitively comprehending the system's behavioral patterns, simplifying the identification and resolution of potential issues. TL enables the application of prior knowledge and expertise from one task to related ones, crucial in software defect prediction. Since new or altered projects may lack adequate historical data for training effective prediction models, TL extracts valuable insights from existing projects to bolster new project defect prediction. This approach not only boosts prediction accuracy but also significantly reduces model training time, enhancing overall efficiency.

A series of rigorous experiments have validated the effectiveness of the proposed method. Comprehensive assessments encompassed test coverage, fault detection rate, algorithm execution time, and stability. The results indicate that the TL-based software DFSM model for cache defect prediction excels in all areas, notably surpassing traditional methods. Specifically, this approach demonstrates clear advantages in test coverage and fault detection rate, indicating a more comprehensive exploration of software functional space and precise identification of potential cache defects.

Nonetheless, despite its notable achievements, this method faces certain challenges and limitations. Firstly, constructing a DFSM model demands specific expertise to ensure accurate state definitions and transition relationships. Secondly, TL's effectiveness relies on the similarity between source and target projects; substantial differences may limit TL's impact. Therefore, careful selection and similarity assessment of source and target projects are crucial for ensuring TL's effectiveness in practical applications.

Future research can explore automated DFSM model construction to minimize human intervention. Additionally, investigating advanced TL strategies can enhance knowledge transfer efficiency and accuracy. Furthermore, integrating other machine learning techniques, such as deep learning and reinforcement learning, could further elevate software defect prediction efficacy. The experimental results show that the TL-based cache defect prediction software DFSM model performs well in terms of test coverage and fault detection rate. This demonstrates the comprehensive exploration ability of this method in the software functional space, as well as its accuracy in identifying potential cache defects. Compared with traditional defect prediction methods, this method has achieved significant advantages in multiple indicators.

V. Conclusion

This article introduces a cache defect prediction method utilizing the software DFSM model and TL. This approach targets the challenge of predicting cache defects in fresh or altered projects by integrating DFSM and TL. Initially, DFSM is utilized to model the cache behavior of the software system. Subsequently, TL facilitates the transfer of knowledge and experience from established projects to newer ones, enabling quick and precise cache defect predictions for these new endeavors.

The test results are impressive, demonstrating strong performance in terms of test coverage, fault detection, algorithm execution time, and stability. As the volume of training data expands, test coverage progressively enhances, affirming the method's efficacy. Additionally, the method's high fault detection rate indicates its proficiency in accurately pinpointing cache faults in software, thereby aiding in the enhancement of software quality and reliability. Moreover, the algorithm exhibits commendable execution time and stability, ensuring predictive accuracy while maintaining efficient operation. Importantly, the prediction performance remains consistent across various experimental settings.

To sum up, the cache defect prediction method of software DFSM model based on TL provides a new and effective means for software quality assurance. Although this method has achieved significant results in multiple aspects, there is still room for further improvement and research. The current DFSM model construction process requires a certain amount of professional knowledge and experience. Future research can explore automated DFSM model construction methods to reduce manual intervention and improve the efficiency and accuracy of model construction. In order to further improve the efficiency and accuracy of transfer learning, future research can explore more advanced transfer learning strategies. Such as transfer learning based on deep learning or transfer learning based on graph neural networks.

References

[1] K. Foss, I. Couckuyt, and A C. Baruta, “Mossoux. Automated software defect detection and identification in vehicular embedded systems,” IEEE Transactions on Intelligent Transportation Systems, vol. 23, pp. 6963-6973, 2021.

[2] Y. Yao, S. Huang, C. Feng, C. Liu, and C. Xu, "CD3T: Cross-project dependency defect detection tool," International Journal of Performability Engineering, vol. 15, pp. 2329, 2019.

[3] R. Wei, Y. Song, Y. Zhang, "Enhanced faster region convolutional neural networks for steel surface defect detection," ISIJ International, vol. 60, pp. 539-545, 2020.

[4] A. K. Gangwar, S. Kumar, "Concept drift in software defect prediction: A method for detecting and handling the drift," ACM Transactions on Internet Technology, vol. 23, pp. 1-28, 2023.

[5] F. Wu, X. Y. Jing, Y. Sun, L. Huang, F. Cui, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," IEEE Transactions on Reliability, vol. 67, pp. 581-597, 2019.

[6] N. Zhang, S. Ying, K. Zhu, and D. Zhu, "Software defect prediction based on stacked sparse denoising autoencoders and enhanced extreme learning machine," IET software, vol. 16, pp. 29-47, 2022.

[7] A. O. Balogun, S. Basri, S. Mahamad, S. J. Abdulkadir, L. F. Capretz, A. A. Imam, and G. Kumar, "Empirical Analysis of rank aggregation-based multi-filter feature selection methods in software defect prediction," Electronics, vol. 10, pp. 179, 2021.

[8] Q. Yubin, C. Xiang, C. Ruijie, X. Ju, and J. Guo, "Active learning using uncertainty sampling and query-by-committee for software defect prediction," International Journal of Performability Engineering, vol. 15, pp. 2701, 2019.

[9] Z. W. Zhang, X. Y. Jing, F. Wu, "Low-rank representation for semi-supervised software defect prediction," IET Software, vol. 12, pp. 527-535, 2018.

[10] L. Gong, S. Jiang, L. Bo, L. Jiang, and J. Qian, "A novel class-imbalance learning approach for both within-project and cross-project defect prediction," IEEE Transactions on Reliability, vol. 69, pp. 40-54, 2020.

[11] H. Wang, W. Zhuang, X. Zhang, "Software defect prediction based on gated hierarchical LSTMs," IEEE Transactions on Reliability, vol. 70, pp. 711-727, 2021.

[12] F. Li, Y. Qu, J. Ji, D. Zhang, and L. Li, "Active learning empirical research on cross-version software defect prediction datasets," International Journal of Performability Engineering, vol. 16, pp. 609, 2020.

[13] D. Wang, H. Yang, H. Zhou, and D. Wang, "Connecting historical changes for cross-version software defect prediction," International Journal of Computer Applications in Technology, vol. 63, pp. 371, 2020.

[14] T. Chakraborty, A. K. Chakraborty, "Hellinger net: A hybrid imbalance learning model to improve software defect prediction," IEEE Transactions on Reliability, vol. 70, pp. 481-494, 2020.

[15] S. Wang, Y. Li, W. Mi, Y. Liu, "Software defect prediction incremental model using ensemble learning," International Journal of Performability Engineering, vol. 16, pp. 1771, 2020.

[16] Q. Yu, S. Jiang, J. Qian, L. Bo, L. Jiang, and G. Zhang, "Process metrics for software defect prediction in object-oriented programs," IET Software, vol. 14, pp. 283-292, 2020.

[17] M. L. Florence, R. Jayanthi, "Improved Bayesian regularisation using neural networks based on feature selection for software defect prediction," International Journal of Computer Applications in Technology, vol. 60, pp. 225, 2019.

[18] H. Tong, B. Liu, S. Wang, "Kernel spectral embedding transfer ensemble for heterogeneous defect prediction," IEEE Transactions on Software Engineering, vol. 47, pp. 1886-1906, 2021.

[19] Q. Song, Y. Guo, M. Shepperd, "A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction," IEEE Transactions on Software Engineering, vol. 45, pp. 1253-1269, 2019.

[20] A. A. Saifan, N. A. Smadi, "Source code-based defect prediction using deep learning and transfer learning," Intelligent Data Analysis, vol. 23, pp. 1243-1269, 2019.

[21] Y. Qu, X. Chen, Y. Zhao, X. Ju, "Impact of hyper parameter optimization for cross-project software defect prediction," International Journal of Performability Engineering, vol. 14, pp. 1291-1299, 2018.

[22] K. Bashir, T. Li, C. W. Yohannese, "An empirical study for enhanced software defect prediction using a learning-based framework," International Journal of Computational Intelligence Systems, vol. 12, pp. 282, 2018.

[23] F. K. El, N. Yevtushenko, A. Saleh, "Incremental and heuristic approaches for deriving adaptive distinguishing test cases for non-deterministic finite-state machines," The Computer Journal, vol. 62, pp. 757-768, 2019.

[24] R. M. Hierons, "Testing from partial finite state machines without harmonised traces," IEEE Transactions on Software Engineering, vol. 43, pp. 1033-1043, 2017.

[25] Y. Shao, B. Liu, S. Wang, G. Li, "A novel software defect prediction based on atomic class-association rule mining," Expert Systems with Applications, vol. 114, pp. 237-254, 2018.