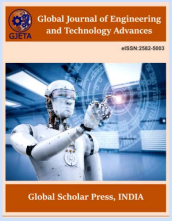
Global Journal of Engineering and Technology Advances
eISSN: 2582-5003
Cross Ref DOI: 10.30574/gjeta
Journal homepage: https://gjeta.com/



(RESEARCH ARTICLE)

Check for updates

Enhancing software quality through predictive analytics: A deep learning approach to defect prediction and prevention

Gopinath Kathiresan *

Department of Software Quality Engineering, Apple Inc, USA.

Global Journal of Engineering and Technology Advances, 2024, 19(01), 212-221

Publication history: Received on 01 March 2024; revised on 10 April 2024; accepted on 13 April 2024

Article DOI: https://doi.org/10.30574/gjeta.2024.19.1.0065

Abstract

In the case of modern applications, software quality is essential because it reflects how reliable, secure, and maintainable the application will be. Manual review and rule-based testing are traditional, but not robust, methods of detecting and preventing defects in software systems, and they tend to fail when the complexity of the software systems grows. This paper investigates how software quality can be improved by early defect prediction and prevention using predictive analytics and deep learning. The base of machine learning powered predictive analytics uses historical defect data to detect patterns that could signal a potential weakness in software. One of the attractive aspects of deep learning is that, due to automatically learning complex representations, it can achieve even higher accuracy in locating defects during software development and automating software quality assurance processes. This paper discusses how different architectures of deep learning such as convolutional neural networks (CNNs), recurrent neural networks (RNNs) and transformers are applied to defect prediction. In addition, it provides some initial insights for implementing deep learning in the domain of software quality assurance, including the lack of data availability, high computational costs and model interpretability. The paper concludes with future research directions to improve defect prediction accuracy, to increase model transparency, and to answer ethical questions such as whether it is ethical to elevate model predictions over human decisions in the practice of software engineering.

Keywords: Software quality; Defect prediction; Predictive analytics; Deep learning; Machine learning; Software engineering; Software defects; Quality assurance; Automated testing; Neural networks

1. Introduction

1.1. The Importance of Software Quality

In the technology environment today, software quality is a vital factor that determines the reliability, security, and efficiency of modern applications. Software high quality leads to better user experience, reduces the costs of maintenance and guarantees compliance with the industry standards. In contrast to poor software quality, which can result in serious consequences such as financial losses, security vulnerabilities, and damage to their reputation, businesses can gain immensely from improved software quality.

* Corresponding author: Gopinath Kathiresan

Figure 1 The Importance of Software Quality in softaware development

Given that software systems are ever more complex, integrated with crucial infrastructures (e.g., finance, healthcare, transportation), and software quality assurance mechanisms will continue to be more required (Tian, 2005). With successful products, user satisfaction and operational efficiency are a big factor while long term costs are critical in the maintenance stage. Software quality assurance is important not only due to the damage it can cause to the software engineering (Lin et al., 2020), but also in terms of business reputation as poor-quality software can lead to security breaches, financial loss, etc.

1.2. Understanding Software Defects and Their Impact

Flaws or errors in a program resulting in the execution of the program producing undesired effects or failing to satisfy user requirements are called defects in software. There are many sources for these defects: logical errors in the code, bad implementation of the algorithm, and insufficient testing procedure. Minor defects cause slow performance issues or something equally inconveniencing, whereas some defects can cause catastrophic failures, like data breach or system crash. For example, software defects in safety critical systems such as medical device or autonomous vehicle may result in life threatening risks (Wahono, 2015).

Software defects do not impact just applications. The financial losses due to large scale system failures have historically been significant. For example, in 2012, in just minutes Knight Capitals' trading system failed through a software bug resulting in a loss of $440 million. As in the case of security protocols defects, sensitive user data has been exposed to cyber-attacks by defects in security protocols (Ghaffarian & Shahriari, 2017).

1.3. The Growing Need for Predictive Analytics in Software Engineering

Current problems of traditional software quality assurance (SQA) techniques, namely those of manual code reviews as well as rule based static analysis, exist when dealing with large scale project and complex and evolving defects remain undetected. Given the growing volume of software code due to less development cycles resulting from agile methodologies, and their need for automated and intelligent defect prediction mechanisms. As machine learning and deep learning techniques show a promise to predict defect-prone segments in the code before they lead to failures (Lessmann et al., 2008), predictive analytics has gained considerable traction.

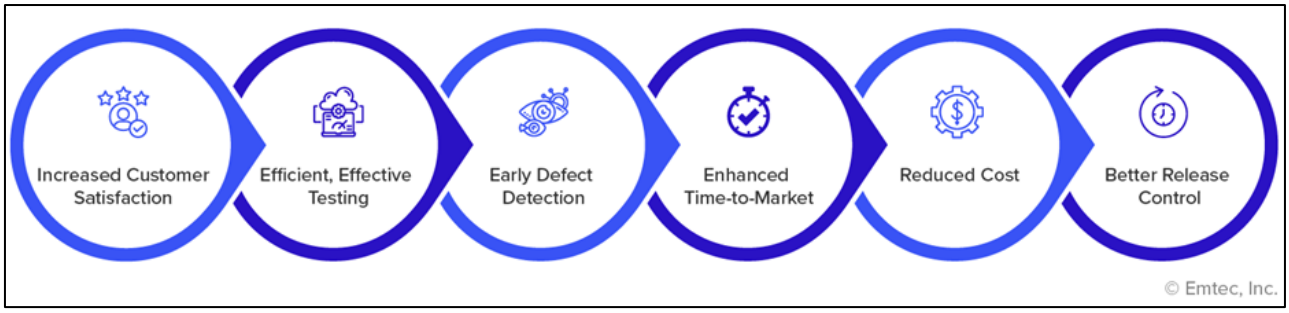


Figure 2 Benefit of Predictive Analytics in Software testing

Software engineering predictive analytics is based on recognizing patterns of historical defect data, training models and predicting defects in new code. These machine learning algorithms include decision trees, support vector machines (SVM) and neural networks used to classify their software modules into defective or non-defective ones. This is done by integrating predictive analytics into the software development lifecycle for organizations to significantly reduce the cost and effort around defect detection and correction in late stages of development (Angelopoulos et al., 2019).

1.4. Introduction to Deep Learning for Defect Prediction and Prevention

Machine learning has gained much attention these days because of its ability to handle complex, high dimensional data, and deep learning is one such subset of machine learning. Unlike traditional machine learning techniques that need handcrafted features, deep learning models learn representations from raw data in an automatic fashion and are very successful in pattern recognition tasks including software defect prediction. Various Neural Network architectures like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Transformers have shown that source code analysis can be done with high accuracy, bug detection is possible and even fault repair work can be suggested (Lin et al., 2020).

Using deep learning, defect prediction accuracy can improve, and software engineering teams can also be automated with a step that is identifying and preventing the possible issue. Large repositories of code qualify as deep learning models, the latter can be trained to learn from past defects to detect similar patterns in new software projects. Furthermore, these models can be plugged into Continuous Integration/Continuous Deployment (CI/CD) pipelines such that software failures do not occur in a production environment (Najafabadi et al., 2015).

1.5. Article Objectives and Structure

The focus of this article is a study of the influence of predictive analytics and deep learning on the improvement of the software quality. It features the in depth of machine learning techniques used in what if software defect prediction and prevention. Section 2 discusses limitations of software defects and traditional quality assurance methods. Section 3 investigates the application of predictive analytics in software engineering, whereas Section 4 studies the problem of deep learning-based defect prediction. In Section 5, we discuss challenges and future research directions, and finally conclude in Section 6 with key insights and recommendation for industry adoption.

2. Understanding Software Defects and Quality Assurance

2.1. Software Defects: Causes and Consequences

Bugs, in other words, are errors, flaws or unexpected actions during the use of a software system. These defects can be characterized in terms of their nature and impact and grouped into several classes. A lack of implementation of an algorithm is another reason for logic errors, as it produces an unintended output. Programming language rules are used incorrectly and hence giving rise to syntax error which prevents program to compile or run. Software is made sensible to potential attacks (such as SQL injections and buffer overflows) that can result in data breaches, or an unauthorized access (Ghaffarian & Shahriari, 2017). Performance defects impact the system's efficiency by dragging performance down, hitting slow execution, high memory usage, as well as causing system hanging or unexpected crashes. In addition, integration errors happen when two modules or one external dependency did not talk to one another because of such version mismatches, bad API treatments, or discordances in data flow (Lin et al., 2020).

Human error is one of the major reasons of the occurrence of software defects. Mistakes that are injected by developers, testers and designers due to fatigue, inexperience, or even misinterpretation of the requirements may be introduced.

Modern applications are also highly software complex comprising of many interconnected components, making the detection of defects quite challenging (Bose & Mahapatra, 2001). Lack of testing valid much problems Postpones the detection until later stages of development or deployment (Kim et al., 2011). Incomplete or ambiguous requirements can make for inconsistencies in implementation resulting in defects which won't show until deployment. Additionally, time constraints and quick development put the focus of teams on speed rather than quality – resulting in defected work that is not even caught by the team. Moreover, the reliance on third party libraries and APIs increases defects risk since external dependencies may include their own vulnerabilities or compatibility issues (Gudivada et al., 2017).

However, software defects can be ranged from gracious inconveniences to cataclysmic failures especially in critical applications. Software bugs in healthcare systems can lead to faulty medical diagnoses or inappropriate treatment plans endangering the lives of the patients (Marimuthu et al., 2018). Failure of radiation therapy software has resulted in incidents of overexposure and severe injuries that are notable examples. Defects have severe economic impact in the financial systems as shown in 2012 Knight Capital, where an automated trading system software bug lost $440 million in less than an hour (Bose & Mahapatra, 2001). In the same sense, mistakes in software in aerospace and automotive industries can be fatal. In 2018 and 2019, two Boeing 737 MAX crashes were related to a defective MCAS software which caused tragic accidents and led to heavy regulatory scrutiny. Moreover, cybersecurity breaches are often caused by software vulnerability, as Equifax data breach in 2017, which exposed the highly sensitive information about over 147 million individuals (Ghaffarian & Shahriari, 2017). These scenarios demonstrate the importance of simulated methods for properly preventing defects and impeccable quality assurance practices in software manufacturing.

2.2. Traditional Approaches to Software Quality Assurance

Software quality assurance (QA) methods that are traditional methodologies have contributed to achieving a degree of software reliability, but have problems associated with them. Manual code review and inspection is one of the oldest approaches, where developers or dedicated review teams would review code line by line to find errors, inconsistency and a potential of security vulnerability. However, code reviews not only improve code maintainability and best practices, but also they are time consuming and rely on human expertise, which limits their effectiveness with large scale project under tight deadlines (Tian, 2005).

Software testing, as another fundamental QA method is made up of static and dynamic techniques. Static testing occurs when the code is statically examined without actually running the program using lint himself, static code analyzers, and formal verification as a means of detecting vulnerabilities early on in development. Unlike in the other case, in the dynamic testing, software is run under controlled conditions to detect defects (Wahono, 2015). Unit testing and integration testing are included in this as well. Unit testing tests out each component on its own, while integration testing makes sure that different modules work together. In system testing, software is tested as a whole, and on the other side there is acceptance testing which will certify that the software enables user defined requirements. Furthermore, security testing targets to find out the weaknesses that would be utilized by the malicious actors (Ghaffarian & Shahriari, 2017).

Traditional QA is effective but is limited. When the complexity of the software is high manual testing and code review are time consuming and subject to human oversight. Some of these issues can be mitigated by automated testing, but it requires the work on setup and a lot of maintenance (Shahbazi & Byun, 2021). Additionally, traditional testing methods implement the role of tests as reactive — the defects are discovered after they have already been created in the system. As a result, this increases the debugging and maintenance costs considerably as there is much higher cost involved in fixing defect in later development stage compared to early stages (Lessmann et al., 2008). With the limitations of traditional QA approaches becoming increasingly apparent, increasingly ahead of the curve, software defect detection and prevention have driven more sophisticated techniques such as machine learning and deep learning to suit the need of software quality assurance (Kibria et al., 2018).

3. Predictive Analytics in Software Quality

3.1. Fundamentals of Predictive Analytics

The term predictive analytics is a data driven approach which uses statistical techniques, machine learning and artificial intelligence to learn from historical data to make predictions about future outcomes. In the domain of software engineering, the use of predictive analytics in detection software defects before the occurrence in the production phase is paramount. With the help of the historical defects' patterns and software metrics, predictive models can predict the probability of defects occurring in the new code (Wahono, 2015) and teams can act to prevent the defect.

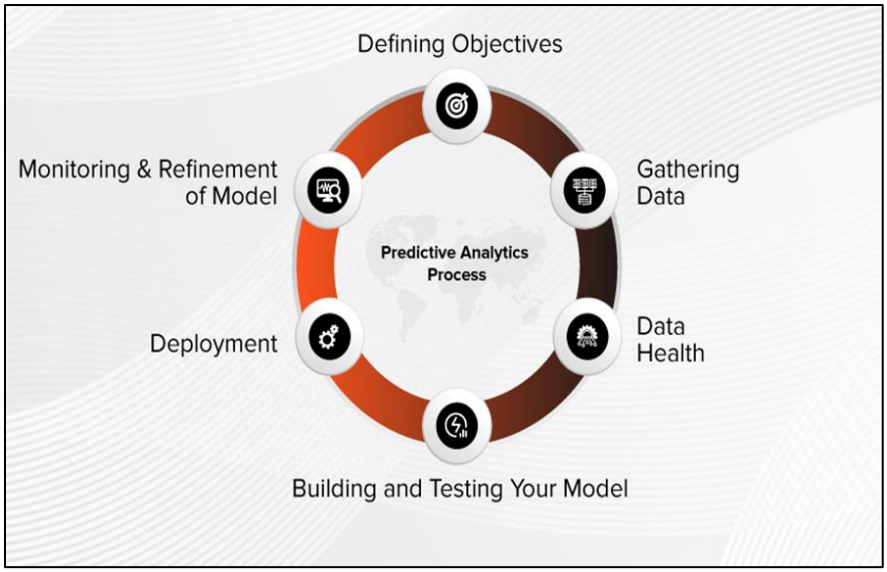


Figure 3 How does predictive analytics work?

There are several common techniques used in predictive analytics for software quality assurance. Software metrics and defect occurrence relationship is analyzed by regression models, such as linear and logistic regression in order to quantify the probability of defects. Software components are classified into decision trees depending upon defect prone attributes for defect prediction and the rules are of interpretable nature. Kelleher et al., (2020) group similar software components using clustering techniques as k-means and hierarchical clustering for the purpose of identifying the patterns of defect prone modules. These techniques aim to facilitate understanding defect trends and thereby improving defect prevention strategies which constitute software engineers.

Integrating predictive analytics in software quality assurance will bring multiple advantages particularly in detecting early defects. In fact, traditional defect detection methods tend to catch the problem at a late stage in the development lifecycle which will definitely result in a delay and a steep cost. Predictive analytics gives developers the capability to forecast defect risk early, which helps target test and take remedial action. Predictive models also allow resources to be allocated and focus efforts in high risk areas of the software thus improving testing and overall software reliability (Lessmann et al., 2008).

3.2. Machine Learning for Software Defect Prediction

Software defect prediction with machine learning has brought such power to emerging because, based on historical data of the software, it is able to analyze complex patterns and make accurate predictions. Defect prediction has mostly been approached through various machine learning techniques, from standard statistical models to state of the art deep learning algorithms. Thus, these techniques are used to classify software components as defect free or defect prone based on processing the software metrics including code complexity, churn rate, and past defect history (Akour & Melhem, 2020).

The type of machine learning fall into two categories namely supervised and unsupervised learning. Training models over known historical occurrences of defect is known as supervised learning. Support vector machines (SVM), random forests, and artificial neural networks are all algorithms that take advantage of this data to predict defect of new code. On the other hand, unsupervised learning finds patterns based on unlabeled data using clustering and anomaly detection techniquies. However, these methods are particularly useful when no labeled defect data is available, forcing these methods to highlight software components with uncharacteristically large sizes, which may indicate a defect (Bose & Mahapatra, 2001).

Software defect prediction using machine learning models however has a few difficulties. A major challenge is quality and availability of the training data. But many organizations don't have well-labelled defect datasets that can be trained well. Furthermore, software projects develop over time; thus, concept drift occurs, i.e., historical defect patterns are no longer suitable for predicting future defect in the future. However, a challenge here is the interpretability and it is unclear how the developers can understand how predictions were made, especially when complex models such as deep

neural networks are used (Gudivada et al., 2017). All these challenges require continuous data refinement, model updating and use of explainable AI techniques to enhance trust in defect prediction systems.

4. Deep Learning for Software Defect Prediction and Prevention

4.1. Deep Learning: An Overview

Machine learning is the subset of deep learning which utilizes both artificial neural networks using many layers to identify complex patterns after the data is gathered. Deep learning differs from traditional machine learning models that are dependent on handcrafted features and takes advantage of learning hierarchical representations on raw data, and is therefore very powerful for dealing with large and complex data sets (Najafabadi et al., 2015). At the same time, this kind of capability, let alone deep learning, is exactly what software defect prediction needs for finding defects in potentially trivial syntax or relationship that more conventional models cannot grasp into.

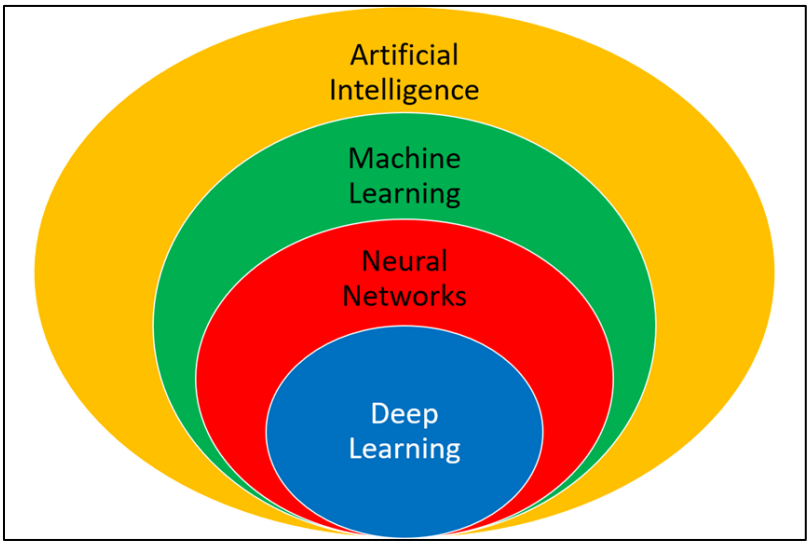


Figure 4 Deep Learning

A number of key architectures have been used in deep learning to address the problem of defect detection. Convolutional Neural Networks (CNNs) emerged as a processing technique first designed for image analysis but now also used for dealing with source code representations, for example, token embeddings and abstract syntax trees. RNNs and its advanced variant, Long Short-Term Memory (LSTM) networks come naturally in order to process sequential data and hence are well suited to analyze the code execution flows and flagging the code execution logic errors (Bouktif et al., 2018). In recent years, Transformers including BERT and GPT based models have demonstrated great promise in natural language processing tasks including defect prediction and automatically code analysis. These architectures are needed to allow deep learning models to consume source code more efficiently than other machine learning techniques.

4.2. Application of Deep Learning in Software Defect Prediction

A software code is processed by deep learning models in order to convert the code to a set of numerical representations that can be analyzed in a way to find patterns that are related to defects. In recent years, word embeddings (e.g., Word2Vec, FastText) as well as graph representations (Lin et al. 2020) have been employed to model code using the semantic and structural properties of it. These models are trained on large datasets of labeled software defects and can learn the artifacts indicative of problems so as to improve the accuracy of defect detection.

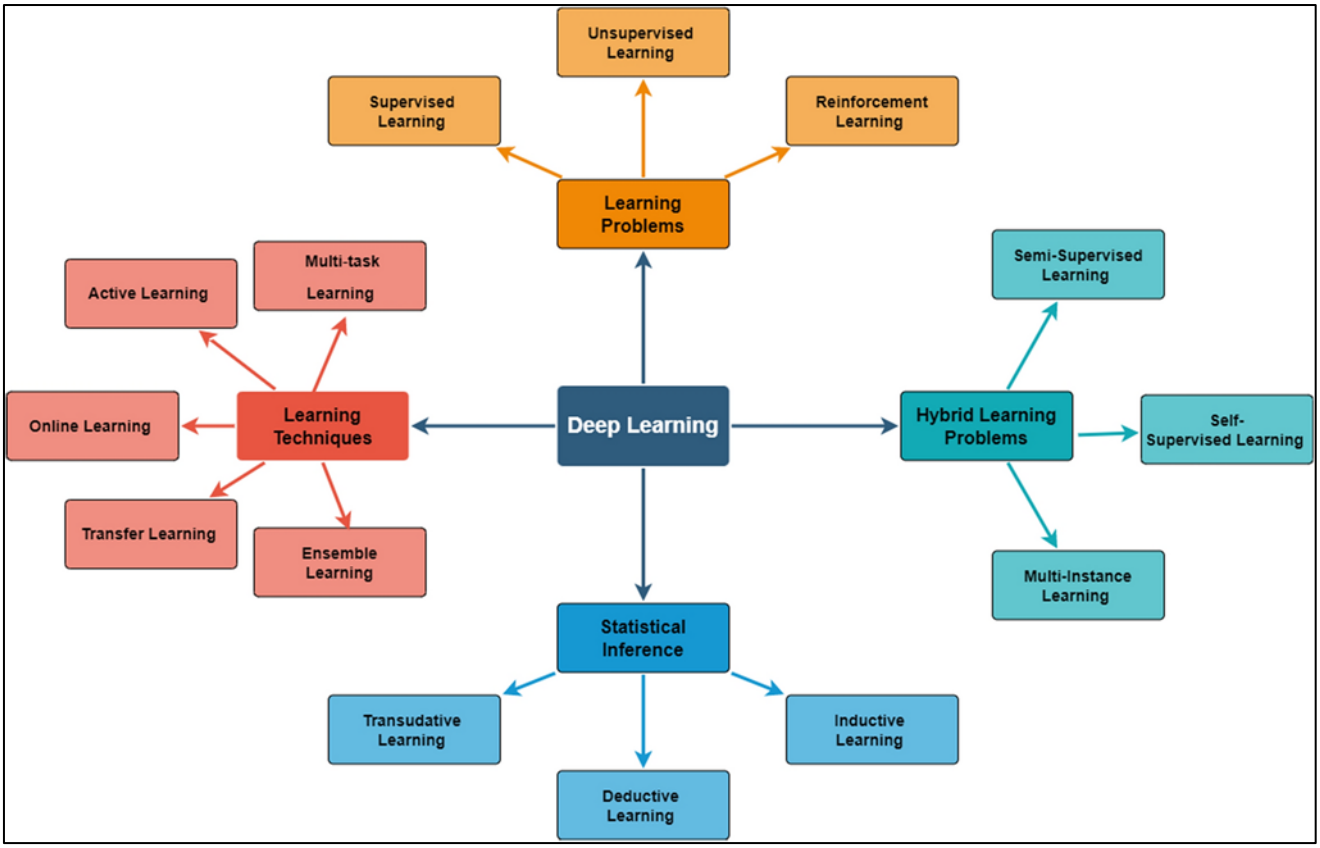


Figure 5 Types of Deep Learning

Automating the review and detection of bugs is one of the key applications of Deep Learning in defect prediction based on textual representations. Static analysis tools as we know them today rely on sets of predefined rules, the applicability of which is usually not very general. In contrast to that, static rules may not be capable of detecting all errors and deep learning models can learn from built up repositories of code to identify defects which may not be caught by static rules. Using these models, the potential issues in the source code repositories can be flagged before these are merged into the main development branch and thus the cost of defects resolution is reduced (Naseer et al., 2018).

Deep learning has been successfully used in several defects' prediction real world applications. As an example, Microsoft and Google have evolved neural model-based models for their development pipeline to identify security issues in open source projects. Just as DeepCode and Codex are the tools that use deep learning techniques to produce intelligent code suggestions to reduce defects for better software quality, there are the other tools like them (Shahbazi & Byun, 2021). Defect detection and code maintainability are just such applications that demonstrate that deep learning can enhance software engineering through automation.

4.3. Preventive Strategies Using Deep Learning

Deep learning models can also be used to prevent defects from lowering software reliability and predict defects. In practice, the defect prone modules can be predicted before deployment. Deep learning models can identify high risk components in a codebase by analyzing past defect data and software metrics, and send developers to test these component (Ayvaz & Alpay, 2021). This targeted approach minimizes defects being introduced into the production. It streamlines the testing process and improves testing efficiency.

The automated refactoring suggestions is another important preventive strategy. If you have high quality codebases available for training, deep learning models, can suggest changes to code restructuring to make it healthier and reduce future defect probability. Additionally, these models can be used in tools to analyze the code complexity which can suggest alternative implementations and could also detect code smells that could cause defects without treated (Akour & Melhem, 2020). The software teams can instill improvement in the code by integrating these suggestions into the development workflow.

Deep learning models can be integrated into CI/CD pipelines for ensuring continuous quality assurance. Deep learning provides an opportunity to solve the problem of continuous monitoring and quality control, where it is achieved through real time defect prediction with automated testing and deployment workflows. This integration enables early defect detection in the software development lifecycle, decreasing the cost for costly rework and improvement of the overall software reliability. AI driven CI/CD strategies have been being adopted by companies like Facebook and Amazon to avoid production defects and increase the performance of the software (Rai, et al., 2021).

5. Challenges and Future Directions

5.1. Challenges in Implementing Deep Learning for Defect Prediction

However, like deep learning, these have the potential to be implemented for defect prediction in software. Data availability and quality is one of the major issues. Labeled defect data is generally sparse and imbalanced, and deep learning models require these models to learn from very large high-quality datasets. However, most software repositories do not provide comprehensive annotation of the defective code and thus have biased models, which are not capable of generalizing on the other projects. Furthermore, software defect datasets may vary in consistency because of the different coding styles, programming languages, and project structures whilst training (Lin et al., 2020).

One important challenge is computational cost as well as the resource constraints when training deep learning models. Deep learning is different from traditional machine learning methods since it relies on a substantial computational power, especially when it is about training large scale neural networks, such as Transformers or LSTMs. These models may not be too easy to deploy efficiently to organizations lacking high performance computing infrastructure. In addition, deep learning solutions may not be very feasible for smaller sized software development teams, since the high cost of GPUs, cloud-based AI service and long training time.

The second challenge that we need to address is model interpretability and trustworthiness. Black box nature of deep learning models makes their internal representation so complex such that one does not easily understand how they achieve the prediction of output values. In software engineering, defect prediction models rely highly on trust, as developers should be able to interpret and receive clear explanation to why mentioning such code segments are defective. If developers don't have sufficient transparency into the models, they may not be willing to trust AI driven recommendation, thereby limiting real world practical adoption of such model (Shahbazi & Byun, 2021).

5.2. Future Research Directions

To solve the current problems, future work should be directed towards further minimizing the errors in defect prediction made by existing deep learning models. An alternative approach is to discover more refined data augmentation algorithms that synthesize synthetic labeled data in order to compensate for the deficiencies in real world defect labels. Furthermore, models can learn better to generalize in different software projects by using transfer learning and domain adaptation strategies which can utilize the knowledge for other tasks that are similar to it (Bouktif et al., 2018). Moreover, hybrid models that combine deep learning with traditional rule-based methods can improve prediction accuracy and reliability.

There have been other important research directions to strive towards enhancing interpretability and explainability of AI models. Through such techniques as attention mechanisms, feature attribution methods (e.g., SHAP, LIME), and visualization tools, the process of how deep learning models discover bugs in the software can be understood. Researches can boost developers' trust in using defect prediction systems, and creating collaborations in AI and human experts in software quality assurance (SQA) (Naseer et al., 2018).

Finally, AI in software quality assurance might be considered from the ethical perspective. With the integration of AI systems into software development processes, it must be examined that concerns about bias, fairness, and accountability, affect the AI systems. Unfair model focus on or ignore a type of defect due to biased training data may impact software reliability. Researchers should investigate approaches intended to audit and mitigate bias in the deep learning models such that defect predictions are fair and stable. Additionally, creating regulatory frameworks and ethical guidelines for the regulation of AI companion software into quality assurance can aid in responsible AI adoption in the industry (Rai et al., 2021).

6. Conclusion

Quality of software has become a key challenge of modern software engineering due to the fact that defects in the software can result in security vulnerabilities, financial loses, and operational failures. Manual code reviews and static analysis are typically ineffective at detecting and preventing defects peacefully. Machine learning models, trained on historical defect data, have risen up as a powerful stream of predicting defect. These models allow us to locate such high-risk areas in software systems before failures occur as a way of obtaining proactive quality assurance with reduced debugging costs.

Further, deep learning automates the prediction and prevention of defect using complex pattern recognition in a cation of software code. CNNs, RNNs, and transformers as neural network architectures offer better accuracy in defects identification and they are useful to be part of a software development pipeline. Integrating deep learning into CI/CD proves beneficial for real time defect monitoring and suggested refactoring in improving the maintainability of software.

One of its strengths is yet another challenge for the implementation of the deep learning to the software defect prediction. That's why many software projects do not have a defect dataset with well labeled defects, and as a result, training data remain the major concerns — in terms of availability and quality. For smaller organizations computational costs of training large neural networks is prohibitive. Moreover, deep learning models are interpreted black boxes, therefore, developers can't rely on predictive accuracy of the model. This can only be achieved by additional research on data augmentation, explainable AI methods, and the ethical consideration of data integrity in AI driven software quality assurance.

Concluding, the future research directions should concentrate on enhancing performance of deep learning models for defect prediction by applying strategies such as transfer learning, hybrid modeling and feature selection. In order to develop a higher developer trust and adoption of model, we need to improve model interpretability using explainable AI frameworks. In addition to these, we need to talk about ethical issues of AI bias and fairness in defect prediction for responsible AI implementations into software engineering. Software quality, defects and system reliability can be improved through the advanced use of predictive analytics and deep learning. There is hope AI quality assurance strategies get readily adopted giving software development a complete course correction from where defect detection and prevention happens more effective, scalable way proactive.

References

[1] Wahono, R. S. (2015). A systematic literature review of software defect prediction. *Journal of software engineering*, *1*(1), 1-16.

[2] Ayvaz, S., & Alpay, K. (2021). Predictive maintenance system for production lines in manufacturing: A machine learning approach using IoT data in real-time. *Expert Systems with Applications*, *173*, 114598.

[3] Akour, M., & Melhem, W. Y. (2020). Software defect prediction using genetic programming and neural networks. In *Deep Learning and Neural Networks: Concepts, Methodologies, Tools, and Applications* (pp. 1577-1597). IGI Global Scientific Publishing.

[4] Shahbazi, Z., & Byun, Y. C. (2021). Integration of blockchain, IoT and machine learning for multistage quality control and enhancing security in smart manufacturing. Sensors, 21(4), 1467.

[5] Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE transactions on software engineering, 34(4), 485-496.

[6] Rai, R., Tiwari, M. K., Ivanov, D., & Dolgui, A. (2021). Machine learning in manufacturing and industry 4.0 applications. International Journal of Production Research, 59(16), 4773-4778.

[7] Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM computing surveys (CSUR), 50(4), 1-36.

[8] Marimuthu, M., Abinaya, M., Hariesh, K. S., Madhankumar, K., & Pavithra, V. (2018). A review on heart disease prediction using machine learning and data analytics approach. International Journal of Computer Applications, 181(18), 20-25.

[9] Gudivada, V., Apon, A., & Ding, J. (2017). Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. International Journal on Advances in Software, 10(1), 1-20.

[10] Kelleher, J. D., Mac Namee, B., & D'arcy, A. (2020). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT press.

[11] Angelopoulos, A., Michailidis, E. T., Nomikos, N., Trakadas, P., Hatziefremidis, A., Voliotis, S., & Zahariadis, T. (2019). Tackling faults in the industry 4.0 era—a survey of machine-learning solutions and key aspects. Sensors, 20(1), 109.

[12] Kibria, M. G., Nguyen, K., Villardi, G. P., Zhao, O., Ishizu, K., & Kojima, F. (2018). Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks. IEEE access, 6, 32328-32338.

[13] Bouktif, S., Fiaz, A., Ouni, A., & Serhani, M. A. (2018). Optimal deep learning lstm model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches. Energies, 11(7), 1636.

[14] Naseer, S., Saleem, Y., Khalid, S., Bashir, M. K., Han, J., Iqbal, M. M., & Han, K. (2018). Enhanced network anomaly detection based on deep neural networks. IEEE access, 6, 48231-48246.

[15] Kim, S., Zhang, H., Wu, R., & Gong, L. (2011, May). Dealing with noise in defect prediction. In Proceedings of the 33rd International Conference on Software Engineering (pp. 481-490).

[16] Bose, I., & Mahapatra, R. K. (2001). Business data mining—a machine learning perspective. Information & management, 39(3), 211-225.

[17] Lin, G., Wen, S., Han, Q. L., Zhang, J., & Xiang, Y. (2020). Software vulnerability detection using deep neural networks: a survey. *Proceedings of the IEEE*, *108*(10), 1825-1848.

[18] Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of big data, 2, 1-21.

[19] Tian, J. (2005). Software quality engine ering: testing, quality assurance, and quantifiable improvement. John Wiley & Sons.

[20] Laith Alzubaidi, Jinshuai Bai, Aiman Al-Sabaawi, Jose Santamaría, A. S. Albahri, Bashar Sami Nayyef Al-dabbagh, Mohammed A. Fadhel, Mohamed Manoufali, Jinglan Zhang, Ali H. Al-Timemy, Ye Duan, Amjed Abdullah, Laith Farhan, Yi Lu, Ashish Gupta, Felix Albu, Amin Abbosh & Yuantong Gu.(2023) A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications. J Big Data 10, 46 (2023). https://doi.org/10.1186/s40537-023-00727-2

[21] Deep Learning (2021) https://apmonitor.com/do/index.php/Main/DeepLearning

[22] https://3scsolution.com/insight/predictive-analytics. (26 Aug 2022) Predictive Analytics: How Can It Improve Business Process Planning?

[23] Bridgenext Think Tank. (11.30.2020). Redefine Software Testing with Predictive analytics. https://www.bridgenext.com/blog/redefine-software-testing-with-predictive-analytics/

[24] Hadi B., (July, 2023). The Importance of Quality Assurance (QA) in Software development. https://www.hb-tech.org/blog/the-importance-of-quality-assurance-qa-in-software-development.