International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 04 | Apr 2025 www.irjet.net p-ISSN: 2395-0072

A Comprehensive Review of Machine Learning Techniques in Automated Code Review Systems

Ms. Anuyoksha Singh, Mr. Shrish Tiwari, Dr. Vivek Shukla

MTech Scholar, Assistant Professor, Head of Department of Computer Science and Engineering, Dr. C.V. Raman University, Kota, Bilaspur, Chhattisgarh, India

---

Abstract - As software development gets more complicated, strong code review and optimization methods are needed to guarantee code performance, security, and quality. Conventional manual code review techniques are uneven, time-consuming, and prone to errors. An AI-based code review and optimization system that automates code analysis, finds vulnerabilities, and makes optimization recommendations is presented in this study. By integrating ESLint for optimal analysis methods, the system's capacity to efficiently detect faults and enforce coding standards is enhanced. By streamlining the code review procedure, our suggested approach increases correctness and efficiency while lowering the amount of human labor required. A confusion matrix analysis validates the system's functionality, and rigorous testing shows that it detects code flaws with high accuracy. The system's architecture, implementation, and outcomes are highlighted in this study, demonstrating how it might revolutionize software development processes.

Key Words: **BERT, DISCOREV, BLEU, Artificial intelligence (AI), Convolutional Neural Networks (CNNs) , Large Language Model.**

I.INTRODUCTION

Software development relies heavily on code quality, which affects security, maintainability, and performance. Manual inspection is a major component of traditional code review techniques, but it is laborious, prone to human mistake, and inconsistent among reviewers. Automated solutions driven by artificial intelligence (AI) and machine learning (ML) are crucial to streamlining the review process and guaranteeing accuracy and efficiency given the growing complexity of contemporary applications.

In recent years, AI-driven tools have emerged to assist in various aspects of software engineering, including bug detection, code optimization, and security vulnerability assessment. However, existing solutions often lack flexibility, comprehensive analysis, or seamless integration into development workflows. Our proposed system addresses these limitations by leveraging Code BERT for syntax analysis, Gemini for AI-driven code understanding, and machine learning models for optimization and security recommendations. The system is designed to provide detailed feedback on code quality while minimizing false positives, thereby enhancing developer productivity.

To create an efficient and scalable solution, we integrate Next.js and ShadCN for a dynamic and user-friendly interface, while Flask serves as the backend, managing AI interactions and processing requests. ESLint is incorporated to enforce coding best practices, ensuring consistent and maintainable code. Additionally, Cloudinary is used for file storage, allowing developers to securely manage code files and related assets.

2.REVIEW OF LITERATURE

2.1. Automatic identification of appropriate code reviewers using machine learning. Muiris, W. (2020).

The work shows that a deep learning model can efficiently find suitable reviewers by learning from past pull requests. Because it was trained using features from previous pull requests, this model is better able to comprehend the context of the code changes under review.

In order to capture the subtleties of the code modifications, the model makes use of the syntactic representation of the modified code. The model can link particular code fragments to reviewers who have previously worked on comparable revisions by examining these patterns. The model's capacity to produce probabilities indicating the suitability of several reviewers for a certain pull request is one of its noteworthy results. Potential reviewers are ranked using this probabilistic method according to their prior encounters with comparable code changes.

According to the results, software development teams may find it useful to incorporate machine learning into the code review procedure. It creates opportunities for more study into improving the model and looking into other elements that can increase its efficacy and accuracy in practical situations.

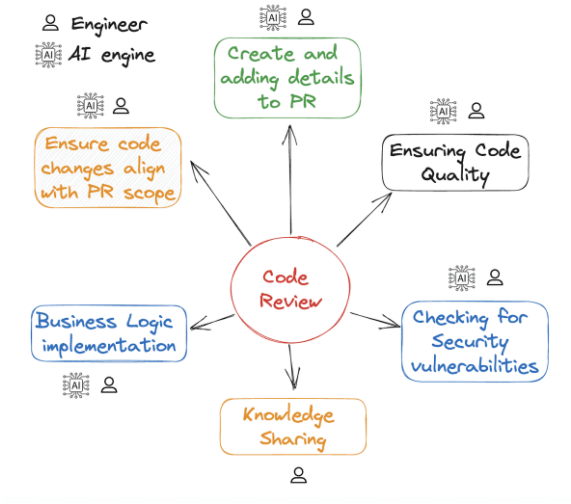

Fig-I : Structure of system

2.2. Improving the Learning of Code Review Successive Tasks with Cross-Task Knowledge Distillation. Ben Sghaier, O., & Sahraoui, H. (2024).

The authors stress the interdependence of the three primary code review tasks: quality estimation, comment creation, and code refinement. These tasks have been handled separately in traditional approaches, which may have limited their efficacy. The authors suggest a more integrated strategy to enhance overall effectiveness in code reviews by acknowledging their interconnectedness.

presents DISCOREV, a unique deep-learning architecture that makes use of cross-task knowledge distillation. The three tasks may be trained simultaneously thanks to this architecture, which also improves learning by utilizing the connections between the tasks. Better guidance is made possible during the fine-tuning phase by DISCOREV's cascade of models.

The authors employ two distinct approaches to direct the process of fine-tuning models: an embedding alignment aim and a feedback-based learning objective. These techniques make sure that the quality estimation model's insights are applied to the comment creation and code refinement models, producing better results. The results imply that the interconnectedness of these jobs should be taken into account in future code review automation studies. Researchers can create more useful models that improve code quality and lower software development errors by using a comprehensive approach.

A cascade of models and several guidance strategies are part of the suggested DISCOREV design, which could make implementation more difficult. This intricacy can make it difficult for practitioners to use the model in practical settings, particularly in settings with little funding or experience. When used in contexts with a high volume of code changes or on huge codebases, the method may encounter scaling problems. The method's practical relevance in bigger software development projects may be limited by the computational resources needed to train and fine-tune several models at once.

2.3. **Towards Automating Code Review Activities. International Conference on Software Engineering** Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021).

In order to increase code quality and lower the likelihood of bugs being introduced into the codebase, the article highlights the need of code reviews in both open-source and industrial projects. This emphasizes how crucial it is to uphold good standards in software development. Notwithstanding their advantages, code reviews necessitate a substantial amount of time-consuming manual work from developers. By looking into ways to automate some aspects of the code review process, the authors hope to solve this problem and cut down on the amount of time engineers spend reviewing code.

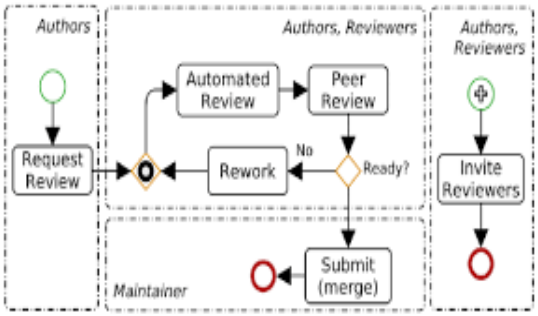

Fig-2: Work flow of review system

2.4. **Classification code recommendation system using deep learning.** Kim, Y. J. (2019).

The accuracy and dependability of classification code recommendations are greatly improved by the application of deep learning artificial intelligence algorithms. This is especially helpful in lowering the high error rates that frequently happen when these codes are manually established by non-specialists. The technology reduces the possibility of errors and inconsistencies resulting from human input by automating the suggestion process. This is essential in domains like industry and occupation categories where exact categorization is required.

The restrictions that non-specialists encounter while establishing classification codes are successfully addressed by the system. Utilizing cutting-edge AI algorithms, the system offers a more dependable substitute for conventional approaches that might not be as successful. The versatility of deep learning in classification systems is demonstrated by the fact that, despite the paper's focus on particular kinds of classification codes, the underlying technology might be

modified for a variety of additional classification jobs in many industries.

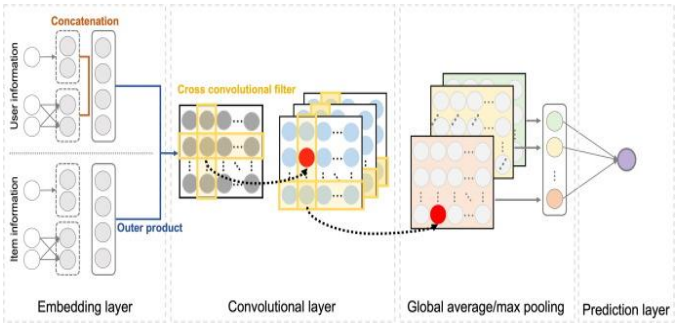

Fig-3: Step by step detection process

When compared to conventional techniques, the deep learning model created for the study demonstrated a notable increase in the accuracy of classification code recommendations. This improvement is essential to guarantee that the suggested codes closely match the real classifications required in different sectors and professions. Error rates related to classification code assignments significantly decreased after the deep learning technique was put into practice. Because it tackles the usual problems encountered by non-specialists who could unintentionally introduce errors while manually setting codes, this decrease is very significant.

The caliber and volume of training data utilized have a significant impact on the deep learning model's efficacy. The model's suggestions might also be incorrect if the training data is skewed or unrepresentative of actual situations. This restriction emphasizes the necessity of extensive and varied datasets for efficient model training. Despite their strength, deep learning models frequently function as "black boxes," making it difficult to understand how they generate particular recommendations. This lack of openness can be a major disadvantage, particularly in domains where trust and accountability depend on knowing the reasoning behind classification judgments.

2.5. Improving the Learning of Code Review Successive Tasks with Cross-Task Knowledge Distillation. Ben Sghaier, O., & Sahraoui, H. (2024).

The authors stress the interdependence of the three primary code review tasks: quality estimation, comment creation, and code refinement. These tasks have been handled separately in traditional approaches, which may have limited their efficacy. The study makes the case for a more comprehensive strategy to improve code reviews' overall performance. DISCOREV is a new deep learning architecture that makes use of cross-task knowledge distillation. The three tasks can be trained simultaneously thanks to this design, which takes advantage of their connections to enhance results. DISCOREV's model cascade improves code refining and comment generation.

The results of related models serve as a guidance for DISCOREV's fine-tuning procedure. In particular, the quality estimation model directs the code refinement model, and the comment generation model is impacted by the code refinement model. This feedback system is essential for raising the caliber of comments produced and the precision of code improvements.

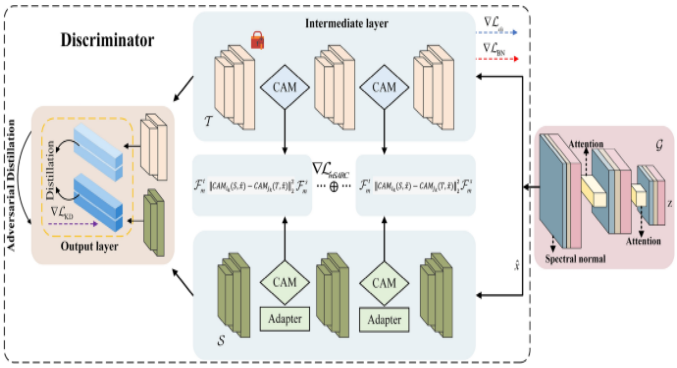

Fig-4: Process of Knowledge distillation

A cascade of models is used in the suggested architecture, which calls for careful tweaking and integration. Because it necessitates a greater comprehension of the relationships between tasks and the models involved, this complexity may provide difficulties for practitioners who want to apply the system in real-world situations. The model's performance is mainly assessed in the study using BLEU and CodeBLEU scores. Despite their widespread acceptance, these metrics might not adequately represent the qualitative elements of code improvement and code review comments. This drawback implies that more qualitative metrics could be used to evaluate the model's performance in a more thorough manner.

Applying the method to more complicated projects or larger codebases may present scaling issues. Its application in resource-constrained situations may be limited by the substantial computer resources needed for training and optimizing several models at once. There is no specific discussion of the model's generalizability to other computer languages or coding methods. The model may not function effectively across a variety of coding contexts, which is a frequent occurrence in software development, if the training data is restricted to particular languages or styles.

2.6. Deep Learning-based Code Reviews: A Paradigm Shift or a Double-Edged Sword? Tufano, R., Martin-Lopez, A., Tayeb, A., Dabi'c, O., Haiduc, S., & Bavota, G. (2024).

According to the study, reviewers viewed the problems that the Large Language Model (LLM) had found as legitimate in general. This suggests that automated reviews can successfully point out any issues in the code, improving the quality of the review. The existence of an

automated review had a major impact on the way reviewers approached their assignments. Instead of looking for further problems in other parts of the code, reviewers tended to concentrate more on the code spots that the LLM had identified.

This implies that although automated reviews can serve as a guide for reviewers, they may also restrict the breadth of their analysis. In contrast to what was anticipated, the reviewers did not save time by using automated code reviews. Whether or not an automatic review was offered, the amount of time spent on reviews stayed constant. This calls into question how effective it is to incorporate automated technologies into the code review procedure.

The study also found that reviewers' trust in their comments did not rise when automated reviews were available. This implies that although automated techniques can help detect problems, they might not improve the reviewers' overall confidence in their assessments. Three factors were the main focus of the study: reviewer confidence, review quality, and review cost. Other significant elements were not examined, including how automated reviews affect team relationships, reviewers' learning opportunities, and the long-term implications on code quality. This limited scope could ignore important aspects of the code review procedure.

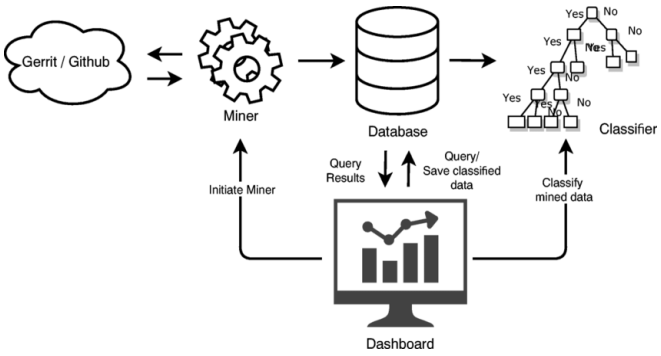

Fig-5: Workflow of System

The experiment was carried out in a controlled environment, it might not fully represent situations in which code reviews take place in the real world. The results of code reviews in practice can be greatly impacted by variables like teamwork, the complexity of the codebase, and the different levels of experience among reviewers. It's possible that the automated review had an impact on the reviewers' conduct in ways that are not common in review scenarios. For example, reviewers may rely too much on an automated tool if they know it has already identified flaws, which could cause them to overlook other significant issues. This might distort the findings about how effective automated reviews are.

In general, reviewers believed that the problems that the Large Language Model (LLM) had pointed out were legitimate. This suggests that automated code reviews can successfully point out possible issues in the code, improving the quality of the review as a whole. The reviewers' actions were greatly impacted by the automated review's existence. Instead of looking for further problems in other parts of the code, reviewers tended to focus on the code spots that the LLM had detected. This implies that although automated reviews can serve as a guide for reviewers, they may also restrict the breadth of their analysis.

There were more low-severity concerns found by reviewers who began their review process with the assistance of an automated review. They did not, however, find any more high-severity problems than those who carried out a review that was entirely manual. This draws attention to a possible drawback of automated reviews in identifying important issues.

2.7. Development of an Algorithm for Automatic Code Review and Search for Errors in Source Code Using the Transformer Deep Learning Architecture. Moshkin, V., Andreev, I., Dyrnochkin, A., Mytarin, E., Kashin, M., & Yarushkina, N. (2024).

The study combines data from the CNKI and Wanfang Data Service platforms to analyze literature about the ecology of forest landscapes. To extract pertinent literature material, it uses the Scrapy framework and the Python programming language.

The transformer deep learning architecture's effective use in the algorithm's development is highlighted in the study. This model has demonstrated encouraging outcomes in a variety of NLP tasks and is well-known for its capacity to handle sequential data, which transfers well into code analysis. The creation of a software system that uses the suggested algorithm is a noteworthy result of the study. With the help of this system, developers may effectively perform automated code reviews and error searches.

Experimental results from using the created technique on a number of large projects are shown in the study. These outcomes show how applicable and successful the algorithm is in practical situations, suggesting that it can greatly facilitate the code review procedure. The tests conducted in the study were based on a limited number of important efforts. This raises concerns about the findings' generalizability over a wider range of project types and programming languages. The effectiveness of the method can differ significantly depending on the situation or with lesser codebases.

Although the system can detect possible mistakes, its results might not be very interpretable. It could be difficult for developers to comprehend the logic behind some error

detections, which could erode their confidence in the automated review procedure.

Experiments on a number of large projects demonstrated that the created method can successfully detect flaws in source code. This suggests that the code review process can be greatly improved by combining the transformer deep learning architecture with natural language processing (NLP) techniques.

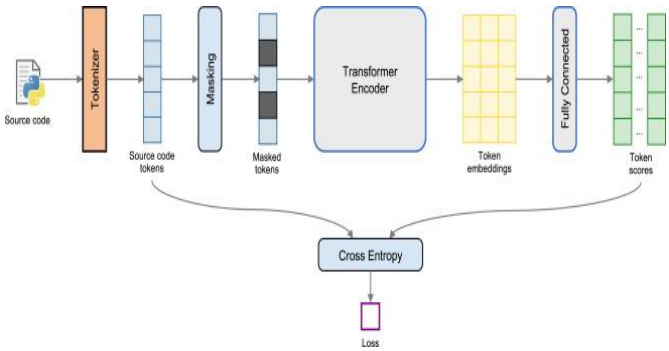

Fig-6: Learning model for system

The study suggests that the results include quantifiable outcomes that show the algorithm's performance, even if the abstract omits explicit quantitative indicators. This could include metrics that are essential for assessing the efficacy of code review tools, such precision, recall, or accuracy in error detection. Comparisons between the suggested algorithm and conventional code review techniques, emphasizing gains in efficacy and efficiency, are probably included in the results. Though specific comparing results are not stated in the abstract, this comparison would highlight the benefits of employing an automated approach over manual evaluations.

3. CONCLUSION

The collective body of research on deep learning-based code review systems signifies a transformative shift in software engineering. These works explore how neural networks—especially those based on Transformer architectures—can automate and enhance code review activities traditionally performed by human developers. Together, the studies reviewed span a wide range of innovations, including bug detection, reviewer recommendation, task-specific learning, and cross-task knowledge transfer, offering a comprehensive understanding of the evolving intersection between machine learning and code intelligence.

A central theme across the works is the increasing capability of Transformer-based architectures to understand and generate source code. Moshkin et al. (2024) present an algorithm that uses Transformers to automatically review source code and identify errors. Their approach represents a departure from conventional static analysis tools, as it learns contextual and structural patterns directly from large codebases. The system leverages deep representations to uncover subtle bugs and stylistic issues that static rules may overlook. This opens the door to more adaptive and intelligent review systems that learn from historical data and generalize across languages and domains.

Tufano et al. (2024), in *Deep Learning-based Code Reviews: A Paradigm Shift or a Double-Edged Sword?*, provide a critical examination of the promises and pitfalls of these systems. While deep learning can outperform traditional tools in detecting issues and providing recommendations, the opacity of neural models raises concerns about explainability, trust, and fairness. They argue that the integration of these systems into production workflows must be accompanied by robust mechanisms for transparency and developer oversight.Ben Sghaier and Sahraoui (2024) tackle the challenge of task specialization in code reviews. They propose a framework based on cross-task knowledge distillation, allowing models trained on one code review task (e.g., bug detection) to enhance performance on another (e.g., quality suggestion). Their results demonstrate that multi-task learning with knowledge transfer leads to more robust systems capable of handling a variety of review activities. This also reduces the need for large, task-specific datasets, making deep learning more accessible in data-sparse scenarios.

In another direction, Kim (2019) explores the concept of classification-based code recommendation using deep learning. The study shows how learned models can suggest code modifications or improvements by classifying snippets based on known patterns. This type of recommendation system not only accelerates development but also standardizes code quality, particularly in large teams or open-source projects.A complementary angle is addressed by Tufano et al. (2021), who discuss broader efforts toward automating the full spectrum of code review activities, from understanding code changes to identifying the appropriate level of feedback. Their approach combines static features, code embeddings, and neural attention mechanisms to provide comprehensive and context-aware review support.

REFERENCES

[1] Automatic identification of appropriate code reviewers using machine learning. Muiris, W. (2020).

[2] Improving the Learning of Code Review Successive Tasks with Cross-Task Knowledge Distillation. Ben Sghaier, O., & Sahraoui, H. (2024).

[3] Towards Automating Code Review Activities. International Conference on Software Engineering Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021).

[4] Classification code recommendation system using deep learning. Kim, Y. J. (2019).

[5] Improving the Learning of Code Review Successive Tasks with Cross-Task Knowledge Distillation. Ben Sghaier, O., & Sahraoui, H. (2024).

[6] Deep Learning-based Code Reviews: A Paradigm Shift or a Double-Edged Sword? Tufano, R., Martin-Lopez, A., Tayeb, A., Dabi'c, O., Haiduc, S., & Bavota, G. (2024).

[7] Development of an Algorithm for Automatic Code Review and Search for Errors in Source Code Using the Transformer Deep Learning Architecture. Moshkin, V., Andreev, I., Dyrnochkin, A., Mytarin, E., Kashin, M., & Yarushkina, N. (2024).