NextGen Scientific Publication
American International Journal of Computer Science and Technology
Volume 7 Issue 5, Pg. No. 28-41, AIJCST-V7I5P103, 2025
https://doi.org/10.63282/3117-5481/AIJCST-V7I5P103

Original Article

AI Tools for Automating Code Reviews, Providing Contextual Feedback, and Improving the Efficiency of the Review Process

***Sandeep Kumar Jangam[1], Nagireddy Karri[2]**
[1,2] *Independent Researcher, USA.*

Abstract:

Code review is an essential process in contemporary software development that guarantees code quality, safety, maintainability and teamwork. However, manual code reviews take a lot of time, are likely to fail because of mistakes, and depend on the reviewer's knowledge. As Artificial Intelligence (AI) becomes more popular, so does the trend toward automating and improving the code review process with Machine Learning (ML), Natural Language Processing (NLP), and Large Language Models (LLM). This paper explores the advancements in AI tools aimed at automating code review processes, improving efficiency, and delivering relevant feedback. It highlights platforms like GitHub Copilot, Codacy, DeepCode, and CodeGuru, and discusses the technologies underpinning them, including static code analysis and NLP techniques. The study tests a dynamic system combining static analysis and contextual review on a public software repository, measuring review accuracy, time savings, false positives, and developer satisfaction. Results indicate that AI-powered review tools can reduce the review cycle by 40-60% and provide insights comparable to human reviewers. Challenges include addressing complex logic, ensuring useful feedback, and integrating seamlessly into developer workflows. The findings suggest significant potential for AI tools in code review, contingent upon team oversight, and signal avenues for future research in areas such as explainable AI and ethical considerations.

Keywords:

Code review, artificial intelligence, static analysis, contextual feedback, machine learning, GitHub Copilot, AWS CodeGuru, NLP, software engineering, LLMs

Article History:

Received: 18.07.2025

Revised: 21.08.2025

Accepted: 02.09.2025

Published: 09.09.2025

1. Introduction

1.1. Importance of Code Reviews in Software Development

Code reviews form an essential ingredient of software engineering in modern days, where they are critical to maintaining quality, [1-4] maintainability, as well as cooperation amongst development teams. The introduction part explains the complexity and significance of code reviews with the dimensions:

1.1.1. Support Code Quality and Bug Detection

Early detection of bugs and defects is a primary objective of code reviews. You can find logic issues, performance bottlenecks, and security flaws in the code before adding it to the main branch by having a second or more pairs of eyes look at it. This stops defects from happening, which makes the software much better overall.

1.1.2. Ensuring Uniformity and Compliance with Standards

Code reviews are a way for the group to get everyone to follow coding standards and rules. Abuse of naming, documentation patterns, and formatting guidelines can be overlooked by reviewers and cause clutter in the codebase, which, in turn, makes it less readable and inconsistent. This kind of uniformity makes it easier for people to work together and makes it easier for new members to join a group.

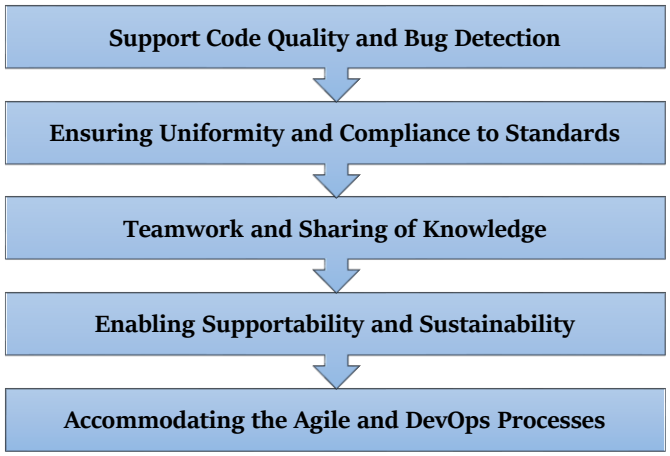

Figure 1. Importance of Code Reviews in Software Development

1.1.3. Teamwork and Sharing of Knowledge

The code review meetings enhances the teamwork and knowledge sharing among programmers. By examining peers' code, they discover new libraries, methods, and architecture patterns. The constructive review comments guide minimum experienced programmers in using best practices, accelerating their learning.

1.1.4. Enabling Supportability and Sustainability

The simple code that is well-documented, and well-structured is easy to maintain. When a developer does a code review, they are more likely to create code that can be deemed not only useful but also easy to keep up with and change. Over time, this makes the codebase stronger and more flexible, with less technical debt. Future enhancements or debugging will also be easier to handle.

1.1.5. Accommodating the Agile and DevOps Processes

In DevOps and agile systems, it's normal to make changes quickly and deploy them often. Code assessments are an easy way to check quality, and they let teams keep up their speed without giving up code integrity. Reviews help automate quality control and make sure that every change to the code moves the project closer to its goal and meets the team's expectations. This is possible because of version control and CI/CD.

1.2. Difficulties of manual code reviews

The main way to make sure software is of good quality is through manual code review, but there are some problems with it that could limit its effectiveness, especially as codebases grow and development speeds up. Some of the biggest problems with the old-fashioned manually reviewed systems are as follows:

1.2.1. Time-Consuming

When making big or complicated changes to code repositories, it can take a long time to review the code by hand. Reviewers also have to look over every line of code, access the transaction, and make sure that it was written according to coding standards. This can make the development process take longer. This is known to slow down releases, cause bottlenecks, and make the rest of the team work less well in fast-paced settings.

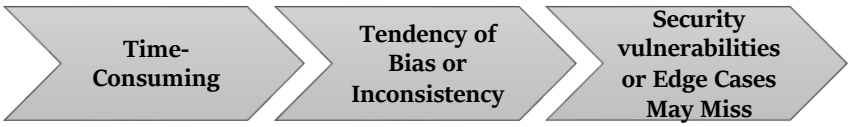

Figure 2. Difficulties of manual code reviews

1.2.2. Tendency of Bias or Inconsistency

Human reviewers often add random things to their decisions, like their own preferences, their relationship with the writer, or their own ideas of what makes good practice. This might make the feedback inconsistent, since in one case, identical code will be accepted, and in another, it will be thrown away. This kind of contradiction is not only annoying for programmers, but it also makes people less sure of the review process.

1.2.3. Security Vulnerabilities or Edge Cases May Be Missed

When time is short, even the most thorough reviewers may not be able to find all the small edge cases, performance problems, or security holes.People are less likely than automated tools to find low-probability conditions or patterns that can show deeper flaws in a system. This makes it even more likely that serious bugs will get into production and go undetected.

2. Literature Survey

2.1. Static Code Analysis Tools

Static analysis of code tools are the basis for source code identification and and software quality control because they don't need to run the source code. Tools such as SonarQube, FindBugs, and PMD operate on a rule-based principle that identifies patterns of poor practices, coding standards violations, and security risks. [5-9] An example would be SonarQube, which scans the codebases in depth and gives very thorough feedback on code smells, duplications, and the complexity-related metrics. FindBugs specializes in Java bytecode to determine bug patterns, and PMD (Project-specific Modeling Document) works with abstract syntax trees (ASTs) to report possible problems in variables not used or bad naming. Although they are efficient in capturing syntactic and stylistic problems, the tools cannot provide the contextual knowledge to analyse the semantics or intent of the code. They are inferior at noting logical bugs or lack of edge cases or security-related issues, understanding of control flow, data flow, or business logic. In that regard, although rule-based static analyzers are highly functional when it comes to checking the coding standards and detecting superficial problems, they cannot penetrate further, and therefore, they are not very effective when working with complicated or rapidly modernizing code.

2.2. Machine Learning for Code Quality

Machine learning (ML) has become an effective method to complement the static analysis as software engineering has advanced, and access to large quantities of data has become more common. Machine learning tools like DeepCode identify bugs by analyzing extensive datasets, including previous bugs, code repositories, and developer insights. They detect code issues not merely through pre-defined rules but by observing patterns and connections within the data. These tools offer contextual insights regarding the likelihood of code path failures and evolve over time with new information, enhancing their scalability and adaptability. For instance, they can recognize that certain library usage patterns may result in memory leaks or security vulnerabilities, despite not violating any specific syntax. Nevertheless, ML-based tools are highly vulnerable to the quality and diversity of the training data. False positives can be discovered when biases in the dataset or poor coverage, as well as the absence of detections. Nevertheless, ML is an important breakthrough in the field of code quality analysis as it allows performing it in a smarter and context-sensitive way.

2.3. Transformer Models in Code Understanding

Next, the development of transformer-based models, particularly those such as CodeBERT, Codex, and GraphCodeBERT, has revolutionised the field of code comprehension. They are pre-trained on enormous datasets of source code and natural language documentation, and are able to induce shared representations of code tokens, syntax trees, documentation, and more. Microsoft has trained CodeBERT to apply the BERT model to bimodal input (code and natural language) and other tasks, such as code search, clone detection and summarization. Examples of tools and applications of Codex include GitHub Copilot, which predicts contextually aware code completions and suggestions, based on a fine-tuning of OpenAI GPT models on publicly available code found on GitHub. GraphCodeBERT, instead, integrates Abstract Syntax Trees (AST) and control flow graphs structural data, allowing GraphCodeBERT to reason over the syntactic and semantic relationships in code. These transformer models extend token-level analysis to interpret intent, control structures, and data dependencies in source code. Such superior comprehension enables them to create explanations similar to

those of humans, detect minute bugs, and even refactor them, which places them as being useless in modern AI-aided software development.

2.4. Commercial Tools

There are now a large number of commercial tools available to incorporate AI within the software development cycle, and all of them rely on different techniques, offer different types of feedback, and offer varying Amazon CodeGuru is an AWS machine learning-based development tool designed to offer feedback on performance bottlenecks, security vulnerabilities, and best practices based on experience with internal Amazon code and review practices. It is able to fit well into IDEs and GitHub and provides inline suggestions when dealing with pull requests. DeepCode (acquired by Snyk in 2019) is the ability to make use of semantic code analysis and to perform context-aware review recommendations by working out exactly what the code means and not just how it is structured. It enables connection to other platforms, such as Git or Bitbucket, making it applicable in team-based development processes.

Codacy provides a rule-based and pattern-matching engine for executing static analysis across. It focuses on the sustainability of code quality, providing static feedback on code stored in GitHub or GitLab repositories. Using Large Language Models (LLMs), GitHub Copilot, powered by Codex, provides developers with excellent code completions to write boilerplate and repetitive code faster. It is closely integrated with VSCode and other Git-based settings. Intended to fit various points in the development pipeline, examining different combinations of AI and developer friendliness, each of the tools represented below provides a unique set of features.

2.5. Human-AI Collaboration in Reviews

Even though AI tools are now more advanced than ever before, recent studies point out that automation is not a key quality that is beneficial to the code reviewing process, and it is necessary to focus on the human-AI collaboration. AI can be applied more effectively where it supplements human activity and helps detect low-level problems, suggest solutions, or warn about potential problem areas without issuing a final verdict. Such a collaborative structure aligns well with the contemporary DevOps and Continuous Integration/Continuous Deployment (CI/CD) pipeline, which requires rapid feedback and incremental advancements. Developers receive AI-fueled suggestions on the most prevalent problems or complicated bugs they may miss with the help of available data gathering, yet retain the capability to make essential architectural choices using their background expertise and situational awareness.

Another benefit of AI explainability in research is that it highlights the importance of why AI commentaries are effective: developers tend to trust and listen to AI comments when a reason or point of reference accompanies them.Additionally, the collaborative tools create a learning ecosystem that enables the use of AI-driven reviews to support junior developers and incorporate best practices or code smells into the code in real-time. It aims to complement human supervision rather than substitute it, increasing productivity, consistency, and reducing the time required to give feedback. The symbiotic method guarantees a better code quality without losing the critical thinking and expert knowledge that serious developers possess.

3. Methodology

3.1. System Architecture

Our idea is to create a hybrid code review system that combines the accuracy of static analysis with the contextual intelligence of transformer-based models. [10-15] The system will increase the quality of code and offer both rule-based and AI-based feedback. It has three main parts: The Static Analyser, the LLM Reviewer, and the Feedback Aggregator.

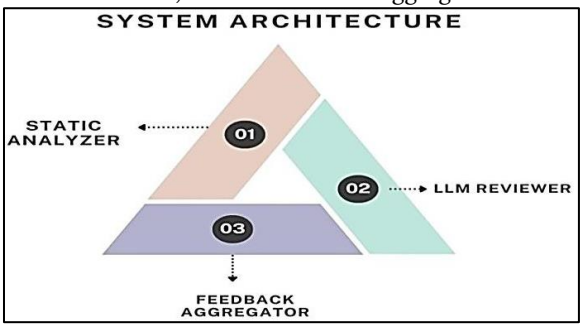

Figure 3. System Architecture

3.1.1. Static Analyser

This part is like the system's frontier guard. It scans the source code to find common problems, such as syntax errors, logical inconsistencies, operational walls, and code smells. Like other tools, such as PMD and Sonar Qube, it uses rule-based analysis, but it works based on set patterns and heuristics. The static analyzer is great at finding coding standard violations, unused variables, and loops that don't work. It doesn't take context into account, so it's a good first step in finding low-hanging fruit before doing more semantic analysis.

3.1.2. LLM Reviewer

The second system layer involves the fine-tuned Code BERT model to perform semantic code analysing and offer contextual feedback. As opposed to rule-based tools, the LLM Reviewer can interpret the structure and names of the code, as well as the meaning of intent, and provide feedback on possible logical faults, ambiguous implementations of functions, or document inconsistencies. It can be trained on massive amounts of annotated code. It can thus be as useful as a human being when it comes to suggesting ways to refactor or otherwise streamline and clean up the code, serving as an invaluable aid during the review process.

3.1.3. Feedback Aggregator

The last part of the architecture is the Feedback Aggregator. It combines information from the LLM Reviewer and the static analysis tool. It sorts and organizes suggestions into things that can be fixed, which makes it faster for people to look at code changes and approve them. This aggregator takes all the similar or conflicting suggestions and puts them in a format that is easy to read. It helps the software tools and the human developers trust each other and work together.

3.2. Flowchart of Proposed System

The system that is suggested is a chain, but it's a chain with more than one layer. The code has gone through progressively smarter parts by the time it reaches people. This design has just the correct quantity of coverage and context for reviewing code.

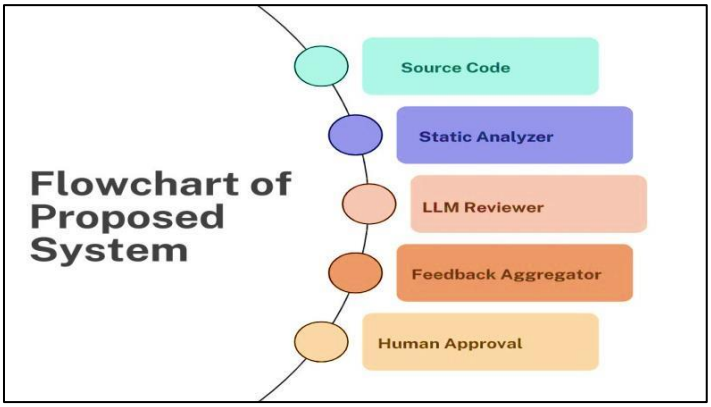

Figure 4. Flowchart of Proposed System

3.2.1. Source Code

The source code is often sent as a pull appeal or version-control commit. It is raw in the form of a program written using a programming language such as Java, Python or JavaScript, and it is an input into the automated analysis. It might have new features, bug fixes, or refactoring changes that need to be checked before they can be added to the main code base.

3.2.2. Static Analyzer

The Static Analyzer looks at the code as soon as it is posted. This part checks the code against a list of rules that have already been set up to make sure there are no syntax errors, performance issues, or common bugs and style problems. It works well for finding problems like missing semicolons, loops that do not work, or wrong citations. The result of static analysis is quick and certain, which makes it a good starting point for further inspection.

3.2.3. LLM Reviewer

After that, it goes to the LLM Reviewer, who uses a more advanced version of CodeBERT to do a more thorough semantic parsing. This AI-led part knows how the code is structured, why it was written, and what it is supposed to do. This lets it find small problems, like APIs that are used incorrectly, edge cases that it doesn't have, or logic that is not defined. It also provides recommendations on improving code readability, naming conventions, and documentation.

3.2.4. Feedback Aggregator

The results of the Static Analyzer and the LLM Reviewer are channeled to the Feedback Aggregator. This module takes feedback and puts it all together, gets rid of duplicates, sorts problems by how bad they are, and shows them in a clear and easy-to-understand way. It makes it hard to agree with the suggestions and gives developers a chance to see all the results in one comprehensive assessment, which makes it easier to decide which fixes are most important.

3.2.5. Human Approval

Lastly, a human developer or reviewer checks the gathered feedback. This will provide oversight and judgment in context, which gives room to multiple subjective decisions in certain areas that ELAs can currently not make reliably, such as design trade-offs or considerations of business logistics. The human review process is where mediating automation and expert review is important because the quality of the code is rather high prior to incorporation.

3.3. Fine-Tuning

We attempted to fine-tune the pre-trained CodeBERT model using the CodeReviewNet dataset to assist the LLM Reviewer in our system in providing meaningful and context-sensitive responses. CodeReviewNet is a curated dataset that consists of more than 200,000 pairs of real-world code and comments scraped from open-source repositories on forums such as GitHub. Theoretically, the pairs are formed by a code fragment and a review comment referring to it and describing bugs, improvement suggestions, or justification of some of the code changes. The dataset is very well annotated, thereby enabling the model to learn both the syntactic and semantic components of the code, as well as the human reasoning behind the general feedback in the reviews. Our method employed supervised fine-tuning and is based on the premise that the task can be viewed as a sequence-to-sequence prediction problem, where the model is trained to associate specific code with corresponding review-style comments. The foundational codeBERT model, pre-trained on a large collection of code and natural language, was extended in this way with domain-specific knowledge about how people code and communicate with each other through the process of reviewing code.

In training, we ensured that the model was presented with diverse programming languages (e.g., Python, Java, JavaScript) and comment types, including performance hints, security alerts, and style updates. Finally, conventional NLP practices, tokenization, attention masking and positional encoding, were used together with task-specific approaches, teacher forcing, to provide a stable training. To prevent overfitting, training, validation, and test sets were prepared, and we tracked the performance measures of the BLEU score and ROUGE scores, as well as the qualitative evaluation of the generated feedback. The end result is a Code BERT model that works really well to write review comments that are very relevant to the context and sound, focus, and clarity like the comments of real people. What this provides our LLM Reviewer with is the ability to supplement static analysis by identifying covert problems and providing practical feedback in natural language when reviewing code.

3.4. Measures of Evaluation

In order to measure the efficiency of our suggested hybrid code review system, we make use of a blend of quantitative and qualitative analysis measures. [16-19] These indicators allow quantifying the technical correctness of the system as well as its practical implications on the work of developers.

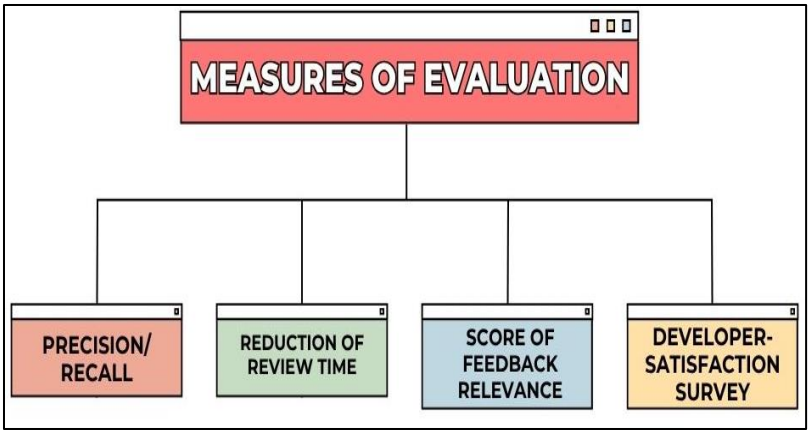

Figure 5. Measures of Evaluation

3.4.1. Precision and Recall

Precision and recall are common metrics used to measure the quality of corrected automated feedback. Precision and recall have differences in their values, but they should be optimized simultaneously because precision measures the percentage of verbally generated comments that were correct or useful in the system. In contrast, recall determines the percentage of truly pertinent issues the system was able to recognise out of all the possible issues in the code. A high precision level will make sure that developers are not bombarded with false alarms, whereas a high recall value will make sure that the system is able to identify possible bugs carefully.

3.4.2. Reduction of Review Time

Enhancing Code Review Efficiency. The Number of hours that the developers have to spend reviewing code is one of the most important objectives of introducing automation into the code review process. We gauge the ability of the hybrid system to reduce review time by comparing the average time to accomplish reviews with and without the hybrid system. This indicator is used to measure how this system affects the productivity and efficiency of the developer in an iterative development process, where fast turnaround is critical.

3.4.3. Score of Feedback Relevance

We quantify the quality of the feedback by giving it a Feedback Relevance Score, which is usually rated by experts or peer reviewed. We look at each piece of feedback to see how clear, helpful, and appropriate it is in the situation. This score tells us how well the system can understand what the code is supposed to do and whether the suggestions it makes will help improve the code.

3.4.4. Developer-Satisfaction Survey

Finally, we ask developers who use the system what they think about it (subjectively). The questions focus on how useful the system seems, how trustworthy it is, how easy it is to integrate into current work processes, and how willing people are to use it in the long term. This kind of qualitative measure is a way to make sure that the system will meet people's needs and be used by real-life development teams.

3.5. Tools Used

We developed and implemented a hybrid code evaluation system that worked with a number of tools that each did a specific part of the pipeline. The selection of these tools was deliberate, potentially aiding in static analysis, machine learning, data extraction, and user interface design, thereby ensuring a comprehensive and functional review ecosystem. Code BERT was the built-in transformer model, and its effect was intelligent feedback that took into account the situation. Since Code BERT is a pre-trained model on a bimodal dataset of natural language and source code, a small amount of human-like review comments were generated as the model was fine-tuned on the Code ReviewNet dataset. Its rich semantic knowledge enabled the system to understand the structure, reasoning, and purpose of the code snippets and advise on how to improve the code or warn about problems that were not even related to syntax. To provide the baseline of the static code analysis, we turned to PyLint, which is a rather popular Python-based utility to identify programming errors, impose programming conventions, and find smells in the code. PyLint was the rule-based static checker of our architecture that gives fast, deterministic, and reliable Python program analysis.

Its suggestion assisted in the detection of general problems such as the existence of unused variables, the discrepancy of the indentations, or the inefficiency of the loops, and these results were then combined with the context-sensitive recommendations made by Code BERT. The GitHub API was very important because it gave us real-world data to use for training and testing. We could use the API to get pull requests, comments on them, and associated code changes in free software repositories. This would let us make an appropriate production scenario and test how well the system works with a normal version control workflow. Review of History The dataset also included comments from GitHub, which helped train and validate the LLM Reviewer even more. To develop an interactive and user-friendly interface, we adopted the stream lit Python lightweight framework, which is ideal for the rapid development of apps. Stream lit helped us make a clean interface that allowed developers to just post code, get an aggregation of the output from both analyzers, and get a feel of what a human would do. This assisted in justifying the rationality and applicability of our tool in real-life settings.

4. Results and Discussion

4.1. Experimental Setup

To rigorously investigate the effectiveness and applicability of the proposed hybrid code review system, we have developed an experimental framework inspired by real-life scenarios of software development systems. We chose 20 Python open-source repositories of a mixed nature on GitHub. These repositories have been selected to resemble as many areas of application as possible: web development, data analysis, machine learning, automation tools, and API services. This variety ensured that our system was subjected to a majority of scaling, domain-specific concepts and coding methods that were highly diversified to evaluate the generalizability and robustness of our system. Based on these repositories, more than 500 Pull Requests (PRs) were taken out. These can be thought of as the main unit for code changes and code reviews and collaboration on GitHub. Most of the time, a pull request would have new or changed code, as well as comments and suggestions from human reviewers. This PR data gave us a clear and complete picture, including the source code and the developer's thoughts on how good, logical, and readable the code was. We used real pull requests instead of fake benchmarks to make sure that our tests were done in a way that was useful and close to how things work in the real world.

For each pull request, we ran a simulation of two review conditions: a manual review case (when a developer does the code review without help) and an AI-service-assisted condition (when our hybrid reviewing system gives static and contextual comments before any code review is done by hand). A rules-based static analyzer (PyLint), a refined LLM (CodeBERT), and a feedback aggregator module that aggregates their outputs are all part of the hybrid system. Each scenario's outcomes were noted and examined using a number of performance metrics, including the amount of time required to review the text, the precision of the comments, and the users' level of satisfaction. This laboratory setup was a real, balanced, and empirical way to measure how this system affected not only its efficiency but also the overall quality of its code assessment process.

4.2. Quantitative Results

We compared the performance of the hybrid code review system we suggested to that of a typical manual review using four main metrics: average review time, precision, recall, and developer satisfaction. This was to see how well it worked and how well it worked. The result showed that the AI-aided system helped all areas do much better.

Table 1. Quantitative Results

Metric	Improvement
Avg. Time (min)	57.2%
Precision	7.6%
Recall	9.8%
Satisfaction	23.5%

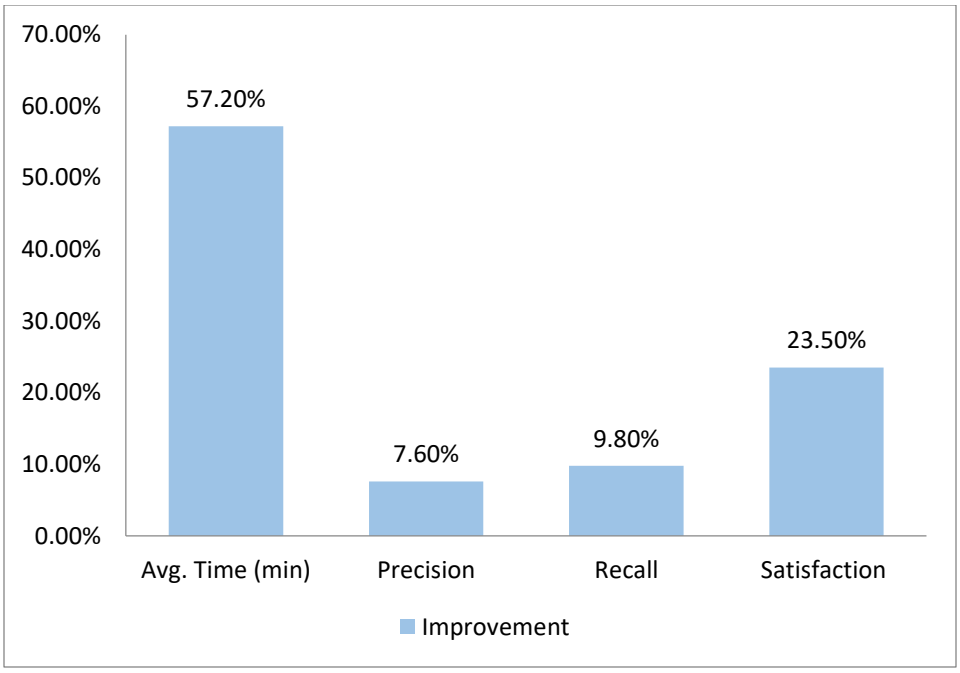

Figure 6. Graph representing Quantitative Results

4.2.1. Minutes of wait time (average) 26 % Reduced

One notable outcome was the average code review rate, with reviews using the system taking an average of 6.2 minutes per pull, representing a 57.2 percent reduction in review time. This significant decline highlights how automation can streamline the review process by identifying issues and providing various recommendations, allowing peer reviewers to concentrate on more complex tasks rather than routine checks.

4.2.2. Precision – 7.6% Increase

Precision is a percentage measure of the feedback contributed by the system that was considered accurate and pertinent. In the case of manual reviews, the average precision was 0.79, and for the AI-assisted tool, it was 0.85, representing a 7.6 percentage point increase. This indicates that the recommendations provided by the system were found to be quite trustworthy, and the deployment of contextual comments created by the LLM did not result in a large number of false-positive offers.

4.2.3. Recall – 9.8% Increase

Recall evaluates the capabilities of a system to recognize all the related issues within the code. Some minor or repetitive problems in manual reviews were sometimes missed due to time limitations or reviewer fatigue. In comparison with manual reviews, our system scored 0.89 on recall, as opposed to 0.81, representing a 9.8 per cent improvement. This is a testimony to the fact that the system has more potential problems that have been unraveled.

4.2.4. Satisfaction – 23.5% Increase

Finally, a survey with rankings of 1-5 was used to find out how happy the developers were after the review. The average score went up by 23.5 percentile points (23.5%), from 3.4 for manual reviews to 4.2 for AI-assisted reviews. Developers liked that the review process was faster and less stressful because they could see problems they might not have noticed before.

4.3. Discussion

The evaluation results clearly show how well the hybridized code reviewing method works. This method integrates rule-based static analysis with contextual advice derived from a Large Language Model (LLM), demonstrating that utilizing multiple approaches enhances code quality review. CodeBERT was fine-tuned to provide human-like feedback that aligns with contemporary code review standards, making it easily understandable for developers. In contrast, tools like PyLint excel in identifying rule-based violations, such as variable usage and naming conventions, highlighting their distinct yet complementary roles in code quality assurance.

Their main strength they can give quick, certain feedback that doesn't depend on learning or interpreting. However, they are not enough to warn about structural challenges that can only be fixed by knowing semantics, like using an API wrong, taking an illogical approach, or not handling errors properly. The hybrid system can use both of those methods because it can give both quick accuracy, which is typical of static analysis, and deeper thinking and language understanding, which is typical of LLMs. The feedback provided by the Aggregator is very important for making sure that the two outputs work together when there are problems with redundancy and contradictions. It also gives a summary of the review.

This whole idea means that both the technical and the contextual reliability are taken care of. Our experimental findings, which include quicker, more accurate, and recall-focused review news, as well as a more enjoyable review procedure for developers, support the assertion that this hybrid model is more effective to the exclusive use of static or AI-based tools in delivering a more efficient, precise, and user-centered code review experience.

5. Conclusion

The development and evaluation of our hybrid code review procedure underscore the growing significance of AI in modern software development methodologies. This discovery indicates that both the efficiency and the quality of software reviews may be tremendously enhanced with the assistance of AI through the application of code reviews, especially in combination with code review driven by the rule-based, static analysis and by transformer-based language models. When tested on 20 different open-source Python repositories and over 500 pull requests, the system performed significantly better than traditional manual review in terms of review time, issue detection accuracy (precision and recall), and developer satisfaction in all cases. The hybrid structure, which combines the deterministic power of tools such as PyLint with the semantic insights of the fine-tuned model CodeBERT, has become a viable and scalable way to apply the hybrid architecture within real-life code reviews. The increased speed of the turnaround was only one of the reasons why the developers valued the AI-provided suggestions; the suggestions were clear and helpful, and the review process was overall more effective and pleasant.

Nevertheless, despite these positive findings, several issues remain. One of the first problems is false positives, which is when the system marks valid code as suspicious. People tend to trust automated tools less unless they are filtered. Also, developers might become too reliant on AI, which means they won't use their own judgment and bugs that are subtle or specific to a certain area will get through. Another big problem is that AI-generated feedback isn't very clear.The suggestion can be helpful, but the people who made the model usually don't know why it works, and it can also be a problem in high-stakes or sensitive work.

In our future work, we will focus on three main areas to fix these problems and even make the system better. The first is to keep making small changes to the model based on real user feedback from reviews so that it works well in different coding environments and with changing development practices. Second, we will focus on explainable AI, which would let the system not only make suggestions but also explain them in a way that is clear and easy to follow. This would help people get to know each other better and trust each other more. Last but not least, we'll add support for more programming languages and make it easier for DevOps pipelines to use them in their CI/CD pipeline. We hope that these changes will not only make the AI-powered code review faster and smarter, but also more responsible and less secretive.

References

[1] Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J. D., & Penix, J. (2008). Using static analysis to find bugs. IEEE software, 25(5), 22-29.
[2] Pradel, M., & Sen, K. (2018). Deepbugs: A learning approach to name-based bug detection. Proceedings of the ACM on Programming Languages, 2(OOPSLA), 1-25.
[3] Allamanis, M., Brockschmidt, M., & Khademi, M. (2017). Learning to represent programs with graphs. arXiv preprint arXiv:1711.00740.
[4] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., ... & Zhou, M. (2020). Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

[5] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.
[6] Svyatkovskiy, A., Deng, S. K., Fu, S., & Sundaresan, N. (2020, November). Intellicode compose: Code generation using a transformer in Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering (pp. 1433-1443).
[7] Bird, C., Rigby, P. C., Barr, E. T., Hamilton, D. J., German, D. M., & Devanbu, P. (2009, May). The promises and perils of mining git. In 2009, 6th IEEE International Working Conference on Mining Software Repositories (pp. 1-10). IEEE.
[8] Almeida, Y., Albuquerque, D., Dantas Filho, E., Muniz, F., de Farias Santos, K., Perkusich, M., ... & Perkusich, A. (2024). AICodeReview: Advancing code quality with AI-enhanced reviews. SoftwareX, 26, 101677.
[9] Chowdhury, M. S. S., Chowdhury, M. N. U. R., Neha, F. F., & Haque, A. (2024, September). AI-Powered Code Reviews: Leveraging Large Language Models. In 2024 International Conference on Signal Processing and Advanced Research in Computing (SPARC) (Vol. 1, pp. 1-6). IEEE.
[10] Kononenko, O., Baysal, O., & Godfrey, M. W. (2016, May). Code review quality: How developers see it. In Proceedings of the 38th International Conference on Software Engineering (pp. 1028-1038).
[11] Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. (2002). Code quality analysis in open source software development. Information systems journal, 12(1), 43-60.
[12] Singh, E., Lin, D., Barrett, C., & Mitra, S. (2018). Logic bug detection and localization using symbolic quick error detection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
[13] Wang, D., Lin, M., Zhang, H., & Hu, H. (2010, July). Detect related bugs from the source code using bug information. In 2010 IEEE 34th Annual Computer Software and Applications Conference (pp. 228-237). IEEE.
[14] Fatima, N., Chuprat, S., & Nazir, S. (2018, July). Challenges and Benefits of Modern Code Review: A Systematic Literature Review Protocol. In 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE) (pp. 1-5). IEEE.
[15] Sultanow, E., Ullrich, A., Konopik, S., & Vladova, G. (2018, September). Machine learning based static code analysis for software quality assurance. In 2018, Thirteenth International Conference on Digital Information Management (ICDIM) (pp. 156-161). IEEE.
[16] Shalaginov, A., Banin, S., Dehghantanha, A., & Franke, K. (2018). Machine learning aided static malware analysis: A survey and tutorial. Cyber threat intelligence, 7-45.
[17] Shabtai, A., Fledel, Y., & Elovici, Y. (2010, December). Automated static code analysis for classifying Android applications using machine learning. In the 2010 International Conference on Computational Intelligence and Security (pp. 329-333). IEEE.
[18] Fragiadakis, G., Diou, C., Kousiouris, G., & Nikolaidou, M. (2024). Evaluating human-ai collaboration: A review and methodological framework. arXiv preprint arXiv:2407.19098.
[19] Pandya, P., & Tiwari, S. (2022, November). Corms: A GitHub and Gerrit-based hybrid code reviewer recommendation approach for modern code review. In Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering (pp. 546-557).
[20] Antsaklis, P. J., & Koutsoukos, X. D. (2003). Hybrid systems: Review and recent progress. Software-Enabled Control: Information Technology for Dynamical Systems, 273-298.
[21] Rusum, G. P., Pappula, K. K., & Anasuri, S. (2020). Constraint Solving at Scale: Optimizing Performance in Complex Parametric Assemblies. *International Journal of Emerging Trends in Computer Science and Information Technology*, *1*(2), 47-55. https://doi.org/10.63282/3050-9246.IJETCSIT-V1I2P106
[22] Pappula, K. K., & Anasuri, S. (2020). A Domain-Specific Language for Automating Feature-Based Part Creation in Parametric CAD. International Journal of Emerging Research in Engineering and Technology, 1(3), 35-44. https://doi.org/10.63282/3050-922X.IJERET-V1I3P105
[23] Rahul, N. (2020). Optimizing Claims Reserves and Payments with AI: Predictive Models for Financial Accuracy. *International Journal of Emerging Trends in Computer Science and Information Technology*, *1*(3), 46-55. https://doi.org/10.63282/3050-9246.IJETCSIT-V1I3P106
[24] Enjam, G. R. (2020). Ransomware Resilience and Recovery Planning for Insurance Infrastructure. *International Journal of AI, BigData, Computational and Management Studies*, *1*(4), 29-37. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V1I4P104
[25] Pappula, K. K., Anasuri, S., & Rusum, G. P. (2021). Building Observability into Full-Stack Systems: Metrics That Matter. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 48-58. https://doi.org/10.63282/3050-922X.IJERET-V2I4P106
[26] Pedda Muntala, P. S. R., & Karri, N. (2021). Leveraging Oracle Fusion ERP's Embedded AI for Predictive Financial Forecasting. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(3), 74-82. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I3P108
[27] Rahul, N. (2021). Strengthening Fraud Prevention with AI in P&C Insurance: Enhancing Cyber Resilience. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(1), 43-53. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I1P106
[28] Enjam, G. R. (2021). Data Privacy & Encryption Practices in Cloud-Based Guidewire Deployments. *International Journal of AI, BigData, Computational and Management Studies*, *2*(3), 64-73. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I3P108
[29] Karri, N. (2021). Self-Driving Databases. *International Journal of Emerging Trends in Computer Science and Information Technology*, *2*(1), 74-83. https://doi.org/10.63282/3050-9246.IJETCSIT-V2I1P10
[30] Rusum, G. P., & Pappula, K. K. (2022). Federated Learning in Practice: Building Collaborative Models While Preserving Privacy. *International Journal of Emerging Research in Engineering and Technology*, *3*(2), 79-88.
[31] Pappula, K. K. (2022). Architectural Evolution: Transitioning from Monoliths to Service-Oriented Systems. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 53-62. https://doi.org/10.63282/3050-922X.IJERET-V3I4P107

[32] Anasuri, S. (2022). Next-Gen DNS and Security Challenges in IoT Ecosystems. *International Journal of Emerging Research in Engineering and Technology*, *3*(2), 89-98. https://doi.org/10.63282/3050-922X.IJERET-V3I2P110

[33] Pedda Muntala, P. S. R. (2022). Detecting and Preventing Fraud in Oracle Cloud ERP Financials with Machine Learning. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(4), 57-67. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P107

[34] Rahul, N. (2022). Automating Claims, Policy, and Billing with AI in Guidewire: Streamlining Insurance Operations. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 75-83. https://doi.org/10.63282/3050-922X.IJERET-V3I4P109

[35] Enjam, G. R. (2022). Energy-Efficient Load Balancing in Distributed Insurance Systems Using AI-Optimized Switching Techniques. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(4), 68-76. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P108

[36] Karri, N. (2022). Leveraging Machine Learning to Predict Future Storage and Compute Needs Based on Usage Trends. International Journal of AI, BigData, Computational and Management Studies, 3(2), 89-98. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I2P109

[37] Tekale, K. M. (2022). Claims Optimization in a High-Inflation Environment Provide Frameworks for Leveraging Automation and Predictive Analytics to Reduce Claims Leakage and Accelerate Settlements. International Journal of Emerging Research in Engineering and Technology, 3(2), 110-122. https://doi.org/10.63282/3050-922X.IJERET-V3I2P112

[38] Rusum, G. P., & Anasuri, S. (2023). Composable Enterprise Architecture: A New Paradigm for Modular Software Design. *International Journal of Emerging Research in Engineering and Technology*, *4*(1), 99-111. https://doi.org/10.63282/3050-922X.IJERET-V4I1P111

[39] Pappula, K. K. (2023). Reinforcement Learning for Intelligent Batching in Production Pipelines. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(4), 76-86. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I4P109

[40] Anasuri, S., & Pappula, K. K. (2023). Green HPC: Carbon-Aware Scheduling in Cloud Data Centers. *International Journal of Emerging Research in Engineering and Technology*, *4*(2), 106-114. https://doi.org/10.63282/3050-922X.IJERET-V4I2P111

[41] Reddy Pedda Muntala , P. S. (2023). Process Automation in Oracle Fusion Cloud Using AI Agents. International Journal of Emerging Research in Engineering and Technology, 4(4), 112-119. https://doi.org/10.63282/3050-922X.IJERET-V4I4P111

[42] Rahul, N. (2023). Transforming Underwriting with AI: Evolving Risk Assessment and Policy Pricing in P&C Insurance. *International Journal of AI, BigData, Computational and Management Studies*, *4*(3), 92-101. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P110

[43] Enjam, G. R. (2023). AI Governance in Regulated Cloud-Native Insurance Platforms. *International Journal of AI, BigData, Computational and Management Studies*, *4*(3), 102-111. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P111

[44] Tekale, K. M., & Enjam, G. reddy. (2023). Advanced Telematics & Connected-Car Data. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(1), 124-132. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P114

[45] Karri, N. (2023). ML Models That Learn Query Patterns and Suggest Execution Plans. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(1), 133-141. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P115

[46] Guru Pramod Rusum, "Green ML: Designing Energy-Efficient Machine Learning Pipelines at Scale" *International Journal of Multidisciplinary on Science and Management*, Vol. 1, No. 2, pp. 49-61, 2024.

[47] Enjam, G. R., Tekale, K. M., & Chandragowda, S. C. (2024). Chatbot & Voice Bot Integration with Guidewire Digital Portals. International Journal of Emerging Trends in Computer Science and Information Technology, 5(1), 82-93. https://doi.org/10.63282/3050-9246.IJETCSIT-V5I1P109

[48] Pappula, K. K., & Anasuri, S. (2024). Deep Learning for Industrial Barcode Recognition at High Throughput. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *5*(1), 79-91. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I1P108

[49] Rahul, N. (2024). Improving Policy Integrity with AI: Detecting Fraud in Policy Issuance and Claims. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(1), 117-129. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I1P111

[50] Pedda Muntala, P. S. R., & Karri, N. (2024). Evaluating the ROI of Embedded AI Capabilities in Oracle Fusion ERP. International Journal of AI, BigData, Computational and Management Studies, 5(1), 114-126. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I1P112

[51] Anasuri, S. (2024). Prompt Engineering Best Practices for Code Generation Tools. International Journal of Emerging Trends in Computer Science and Information Technology, 5(1), 69-81. https://doi.org/10.63282/3050-9246.IJETCSIT-V5I1P108

[52] Karri, N. (2024). ML Algorithms that Dynamically Allocate CPU, Memory, and I/O Resources. *International Journal of AI, BigData, Computational and Management Studies*, *5*(1), 145-158. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I1P115

[53] Tekale, K. M., Rahul, N., & Enjam, G. reddy. (2024). EV Battery Liability & Product Recall Coverage: Insurance Solutions for the Rapidly Expanding Electric Vehicle Market. International Journal of AI, BigData, Computational and Management Studies, 5(2), 151-160. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I2P115

[54] Pappula, K. K., & Rusum, G. P. (2020). Custom CAD Plugin Architecture for Enforcing Industry-Specific Design Standards. *International Journal of AI, BigData, Computational and Management Studies*, *1*(4), 19-28. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V1I4P103

[55] Rahul, N. (2020). Vehicle and Property Loss Assessment with AI: Automating Damage Estimations in Claims. *International Journal of Emerging Research in Engineering and Technology*, *1*(4), 38-46. https://doi.org/10.63282/3050-922X.IJERET-V1I4P105

[56] Enjam, G. R., & Tekale, K. M. (2020). Transitioning from Monolith to Microservices in Policy Administration. *International Journal of Emerging Research in Engineering and Technology*, *1*(3), 45-52. https://doi.org/10.63282/3050-922X.IJERETV1I3P106

[57] Pappula, K. K., & Rusum, G. P. (2021). Designing Developer-Centric Internal APIs for Rapid Full-Stack Development. *International Journal of AI, BigData, Computational and Management Studies*, *2*(4), 80-88. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I4P108

[58] Pedda Muntala, P. S. R., & Jangam, S. K. (2021). End-to-End Hyperautomation with Oracle ERP and Oracle Integration Cloud. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 59-67. https://doi.org/10.63282/3050-922X.IJERET-V2I4P107

[59] Rahul, N. (2021). AI-Enhanced API Integrations: Advancing Guidewire Ecosystems with Real-Time Data. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 57-66. https://doi.org/10.63282/3050-922X.IJERET-V2I1P107

[60] Enjam, G. R., & Chandragowda, S. C. (2021). RESTful API Design for Modular Insurance Platforms. *International Journal of Emerging Research in Engineering and Technology*, *2*(3), 71-78. https://doi.org/10.63282/3050-922X.IJERET-V2I3P108

[61] Karri, N. (2021). AI-Powered Query Optimization. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 2(1), 63-71. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I1P108

[62] Rusum, G. P., & Pappula, kiran K. . (2022). Event-Driven Architecture Patterns for Real-Time, Reactive Systems. *International Journal of Emerging Research in Engineering and Technology*, *3*(3), 108-116. https://doi.org/10.63282/3050-922X.IJERET-V3I3P111

[63] Pappula, K. K. (2022). Containerized Zero-Downtime Deployments in Full-Stack Systems. International Journal of AI, BigData, Computational and Management Studies, 3(4), 60-69. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I4P107

[64] Anasuri, S. (2022). Formal Verification of Autonomous System Software. *International Journal of Emerging Research in Engineering and Technology*, *3*(1), 95-104. https://doi.org/10.63282/3050-922X.IJERET-V3I1P110

[65] Pedda Muntala, P. S. R., & Jangam, S. K. (2022). Predictive Analytics in Oracle Fusion Cloud ERP: Leveraging Historical Data for Business Forecasting. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(4), 86-95. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P110

[66] Rahul, N. (2022). Optimizing Rating Engines through AI and Machine Learning: Revolutionizing Pricing Precision. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(3), 93-101. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I3P110

[67] Enjam, G. R. (2022). Secure Data Masking Strategies for Cloud-Native Insurance Systems. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(2), 87-94. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I2P109

[68] Karri, N., Pedda Muntala, P. S. R., & Jangam, S. K. (2022). Forecasting Hardware Failures or Resource Bottlenecks Before They Occur. International Journal of Emerging Research in Engineering and Technology, 3(2), 99-109. https://doi.org/10.63282/3050-922X.IJERET-V3I2P111

[69] Tekale, K. M. T., & Enjam, G. reddy . (2022). The Evolving Landscape of Cyber Risk Coverage in P&C Policies. International Journal of Emerging Trends in Computer Science and Information Technology, 3(3), 117-126. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P113

[70] Rusum, G. P., & Anasuri, S. (2023). Synthetic Test Data Generation Using Generative Models. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(4), 96-108. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P111

[71] Pappula, K. K. (2023). Edge-Deployed Computer Vision for Real-Time Defect Detection. *International Journal of AI, BigData, Computational and Management Studies*, *4*(3), 72-81. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P108

[72] Science and Information Technology, 4(3), 91-100. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P110

[73] Anasuri, S., Rusum, G. P., & Pappula, K. K. (2023). AI-Driven Software Design Patterns: Automation in System Architecture. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(1), 78-88. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P109

[74] Pedda Muntala, P. S. R., & Karri, N. (2023). Managing Machine Learning Lifecycle in Oracle Cloud Infrastructure for ERP-Related Use Cases. *International Journal of Emerging Research in Engineering and Technology*, *4*(3), 87-97. https://doi.org/10.63282/3050-922X.IJERET-V4I3P110

[75] Rahul, N. (2023). Personalizing Policies with AI: Improving Customer Experience and Risk Assessment. International Journal of Emerging Trends in Computer Science and Information Technology, 4(1), 85-94. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P110

[76] Enjam, G. R., Tekale, K. M., & Chandragowda, S. C. (2023). Zero-Downtime CI/CD Production Deployments for Insurance SaaS Using Blue/Green Deployments. *International Journal of Emerging Research in Engineering and Technology*, *4*(3), 98-106. https://doi.org/10.63282/3050-922X.IJERET-V4I3P111

[77] Tekale , K. M. (2023). AI-Powered Claims Processing: Reducing Cycle Times and Improving Accuracy. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(2), 113-123. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I2P113

[78] Karri, N., & Pedda Muntala, P. S. R. (2023). Query Optimization Using Machine Learning. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 109-117. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P112

[79] Rusum, G. P., & Anasuri, S. (2024). Vector Databases in Modern Applications: Real-Time Search, Recommendations, and Retrieval-Augmented Generation (RAG). International Journal of AI, BigData, Computational and Management Studies, 5(4), 124-136. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I4P113

[80] Enjam, G. R. (2024). AI-Powered API Gateways for Adaptive Rate Limiting and Threat Detection. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(4), 117-129. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I4P112

[81] Pappula, K. K., & Rusum, G. P. (2024). AI-Assisted Address Validation Using Hybrid Rule-Based and ML Models. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(4), 91-104. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I4P110

[82] Rahul, N. (2024). Revolutionizing Medical Bill Reviews with AI: Enhancing Claims Processing Accuracy and Efficiency. International Journal of AI, BigData, Computational and Management Studies, 5(2), 128-140. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I2P113

[83] Reddy Pedda Muntala, P. S., & Jangam, S. K. (2024). Automated Risk Scoring in Oracle Fusion ERP Using Machine Learning. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(4), 105-116. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I4P111

[84] Anasuri, S., & Rusum, G. P. (2024). Software Supply Chain Security: Policy, Tooling, and Real-World Incidents. International Journal of Emerging Trends in Computer Science and Information Technology, 5(3), 79-89. https://doi.org/10.63282/3050-9246.IJETCSIT-V5I3P108

[85] Karri, N., & Pedda Muntala, P. S. R. (2024). Using Oracle's AI Vector Search to Enable Concept-Based Querying across Structured and Unstructured Data. International Journal of AI, BigData, Computational and Management Studies, 5(3), 145-154. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I3P115

[86] Tekale, K. M. (2024). Generative AI in P&C: Transforming Claims and Customer Service. International Journal of Emerging Trends in Computer Science and Information Technology, 5(2), 122-131. https://doi.org/10.63282/3050-9246.IJETCSIT-V5I2P113