iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79602

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

AI-Based Intelligent Code Review System for Students and IT Professionals

M.Vigneshwaran, S.Vinith, S.Bahadoorsha Mohamed, G.Gopal
Artificial Intelligence and Data Science M.I.E.T Engineering College Tiruchirappalli, India

Abstract: ***Modern software development demands rigorous code quality assurance, yet conventional manual review methods remain slow, inconsistent, and reliant on reviewer expertise. This paper presents an AI-Based Intelligence Code Review System that automates source code analysis through the integration of Artificial Intelligence (AI), Machine Learning (ML), and Natural Language Processing (NLP). The system detects syntax errors, logical defects, and coding standard violations while delivering real-time, context-aware feedback to users. A multi-module architecture encompasses an AI code editor, bug detection engine, performance and security analysis components, and an adaptive learning assistant. Evaluation against manual review and static analysis baselines demonstrates that the proposed system achieves 94% accuracy, 92% precision, 91% recall, and an F1-score of 91%, outperforming existing approaches. The platform serves both educational and professional contexts, bridging the gap between automated error detection and continuous skill development.***

Keywords: Code Review, Artificial Intelligence, Machine Learning, Natural Language Processing, Static Analysis, Software Quality, Adaptive Learning, Bug Detection.

I. INTRODUCTION

The integrity and maintainability of software systems are deeply tied to the quality of their underlying code. Code review is a well-established practice for identifying defects before they reach production; however, traditional peer-review workflows impose substantial time overheads and yield outcomes that vary with the reviewer's experience and availability [1].

The rapid evolution of Artificial Intelligence (AI), Machine Learning (ML), and Natural Language Processing (NLP) has opened new avenues for automating repetitive and expertise-dependent engineering tasks. Contemporary large language models (LLMs) and transformer-based architectures exhibit a strong capacity to understand programming syntax, semantics, and style, making them well-suited for intelligent code analysis [2][3].

This paper describes an AI-Based Intelligence Code Review System designed to address the shortcomings of both manual review and rule-based static analysis. The platform supports multiple programming languages, integrates with standard development workflows, and combines error detection with personalised learning support — making it equally applicable in academic and industrial settings

II. LITERATURE SURVEY

Lee and Joe [4] developed a GPT-4o-powered review framework incorporating a Review Necessity Chain (RNC) to minimise unnecessary token processing. The system substantially improved error detection rates versus conventional tools, though its dependency on large proprietary LLMs raises deployment cost concerns.

Borrowable et al. [5] introduced an AI-enhanced code editor built on React.js and Node.js that offers real-time analysis, syntax highlighting, and complexity evaluation across more than fifty languages. While effective for collaborative settings, the system's reliance on external APIs creates connectivity dependencies.

Adapa et al. [6] proposed a GitHub-integrated pull request review assistant leveraging the Falcon-40B generative model. Triggered via webhooks, the system automatically annotates formatting violations and assigns reviewers. Its ROUGE-based evaluation demonstrated approximately 0.91 accuracy, though functional logic assessment remained a stated future goal.

Nimraka et al. [7] advanced the state of the art with a multi-agent agentic system employing Graph Neural Networks (GNNs) for dependency analysis. Agents specialised in security, performance, and quality worked in concert; however, the architectural complexity and resource demands limit adoption in resource-constrained environments.

Xuan and Lee [8] targeted beginner programmers with a classification-based bug detection GUI, demonstrating the value of accessible interfaces. Halvadia and Anvik [9] addressed reviewer assignment through data-driven matching of code changes to domain experts using historical contribution patterns.

Preethi et al. [10] presented CodeBlizz, an IDE plugin combining CodeBERT, GPT, and T5 models for in-editor feedback, reducing context-switching for developers. Collectively, these works highlight two persistent limitations: high computational cost and insufficient integration of learning support alongside automated review.

III. PROPOSED SYSTEM

The proposed AI-Based Intelligence Code Review System is a full-stack, multi-module platform designed to automate source code analysis for both students and IT professionals. The system accepts code submissions through a web-based interface and processes them through a six-stage analysis pipeline using AI, ML, and NLP techniques.

The six-stage pipeline operates as follows: (1) Preprocessing strips comments and normalises whitespace; (2) Tokenisation decomposes source text into syntactic units such as keywords, operators, and identifiers; (3) Syntax Analysis compiles code in a sandboxed environment to detect parse-level errors; (4) Semantic Analysis inspects control-flow constructs for logical anomalies including infinite loops and misused operators; (5) Performance Evaluation identifies nested loop patterns and redundant computations; (6) AI-Driven Suggestion Generation invokes the language model to produce natural-language corrections tailored to detected defects.

A Review Necessity Chain (RNC) filters submissions before LLM invocation, classifying code as requiring review (yes), already correct (no_correct), or trivially simple (no_meaningless). This mechanism prevents unnecessary API calls in large-scale deployments and achieved an 86% blocking rate for meaningless submissions against state-of-the-art baselines [1].

IV. SYSTEM ARCHITECTURE

The system adopts a layered modular architecture consisting of four primary tiers. The Frontend, implemented in React.js, Next.js, and Tailwind CSS, provides the code editor interface, dashboard, and analytics visualisation. The Backend, built with Node.js and FastAPI (Python), handles API orchestration and session management. The AI Engine integrates the OpenAI API and custom ML models for code analysis, bug detection, and feedback generation. The Database tier uses MongoDB and MySQL for persistent storage of submissions, user profiles, and analytics. AWS cloud infrastructure provides deployment scalability.

The user submits code through the editor interface. The backend forwards the submission to the AI engine, which executes the six-stage analysis pipeline. Results — including error annotations, optimization suggestions, and complexity metrics — are returned to the frontend and stored in the database for longitudinal performance tracking.

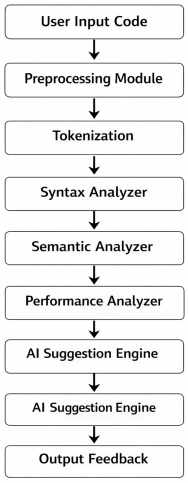


Fig. 1. Architecture of AI-Based Code Review System

V. MODULES DESCRIPTION

A. AI Code Editor and Review Module

The core module provides a Monaco-based code editor with real-time AI-powered analysis. Upon submission, the system returns line-specific error annotations, inline review comments, and suggested corrections within seconds. The editor supports multiple programming languages and executes code in a secure sandboxed environment.

B. AI Mentorship Assistant (Mind Arc Module)

The Mind Arc Module provides a conversational AI mentor interface. Users interact with the assistant in natural language to receive concept explanations, algorithm walkthroughs, and guided debugging support. The assistant adapts responses based on the user's proficiency level and prior interaction history.

C. Performance Analytics Module

The analytics module tracks key metrics including number of problems solved, accuracy rate, domain-wise proficiency, and coding streak over time. Insights are presented through an interactive dashboard, enabling users to identify skill gaps and monitor progress systematically.

D. Learning Modules and Challenges

Structured learning paths covering Data Structures, Algorithms, System Design, and core CS subjects guide users from foundational to advanced topics. Daily and weekly coding challenges encourage consistent engagement. The system records challenge completion and integrates results into the performance analytics dashboard.

E. Notification and Career Roadmap

Real-time notifications alert users to challenge deadlines, review completion, and performance milestones. The Career Roadmap module suggests personalised learning trajectories based on user-defined professional goals, mapping required competencies to structured learning sequences within the platform.

VI. MODEL

The AI-Based Intelligence Code Review System employs a modular architecture that integrates multiple AI and ML components to deliver comprehensive code analysis. The system is built on a layered model consisting of four primary tiers

A. Frontend Layer

The user-facing layer is implemented using React.js, Next.js, and Tailwind CSS. It provides the Monaco-based code editor interface, performance dashboard, and analytics visualisation. The frontend communicates asynchronously with the backend via RESTful APIs and WebSocket connections to deliver real-time feedback.

B. Backend Layer

The backend is built with Node.js and FastAPI (Python) for API orchestration, session management, and secure routing. It acts as an intermediary between the frontend interface and the AI engine, preprocessing submitted code before forwarding it to the analysis pipeline.

C. AI Engine

The AI Engine integrates the OpenAI API and custom ML models for code analysis, bug detection, and feedback generation. It executes the six-stage pipeline — from preprocessing and tokenisation through to syntax and semantic analysis, performance evaluation, and AI-driven suggestion generation. A Review Necessity Chain (RNC) filters submissions prior to LLM invocation to suppress unnecessary API calls.

D. Database Layer

The database tier uses MongoDB and MySQL for persistent storage of code submissions, user profiles, historical performance data, and analytics records. AWS cloud infrastructure provides deployment scalability, ensuring robust performance under concurrent workloads.

VII. METHODOLOGY

The methodology of the proposed system follows a structured pipeline designed to automate and enhance the code review process. The system accepts source code through a web-based editor interface and subjects it to a sequential, multi-stage analysis workflow.

A. Code Submission and Preprocessing

Users submit source code through the Monaco-based editor. The preprocessing stage strips inline comments, normalises whitespace, and standardises indentation to prepare a clean input for subsequent analysis stages. This ensures that the AI engine receives consistent, well-formed code regardless of the original formatting style.

B. Tokenisation and Syntax Analysis

The preprocessed source text is decomposed into syntactic units — keywords, operators, identifiers, and literals — through tokenisation. Syntax analysis then compiles the tokenised code within a secure sandboxed environment to detect parse-level errors such as missing brackets, undefined variables, and type mismatches.

C. Semantic and Performance Analysis

Semantic analysis inspects control-flow constructs for logical anomalies including infinite loops, unreachable code, and misused operators. Performance evaluation identifies nested loop patterns, redundant computations, and suboptimal data structure usage, providing Big-O complexity estimates and recommending algorithmic alternatives where appropriate.

D. Security Analysis

A parallel security module evaluates code for injection-prone constructs, unsafe input handling, hardcoded credentials, and insecure API calls. Detected vulnerabilities are annotated with severity ratings and remediation suggestions.

E. AI-Driven Feedback Generation

Submissions passing the Review Necessity Chain are forwarded to the language model, which generates natural-language corrections, optimised code snippets, and explanatory comments tailored to the detected defects. The feedback is structured to serve both novice learners and experienced developers by adapting the level of explanation to the user's proficiency profile.

F. Results Delivery and Storage

Analysis results — including error annotations, optimisation suggestions, complexity metrics, and security alerts — are returned to the frontend for immediate display and stored in the database for longitudinal performance tracking. The adaptive learning assistant uses historical records to personalise future feedback and learning recommendations.

VIII. ALGORITHM DESIGN

The core algorithm of the proposed AI-Based Intelligence Code Review System follows an eight-step sequential processing pipeline designed to ensure efficient and accurate code analysis.

Step1: Accept user input — Receive source code submission through the web-based editor interface.
Step2: Preprocess code — Strip comments, normalise whitespace, and standardise formatting.
Step 3: Tokenize input — Decompose source text into syntactic units including keywords, operators, and identifiers.
Step4: Perform syntax check — Compile tokenised code in a sandboxed environment to detect parse-level errors.
Step 5: Analyze semantics — Inspect control-flow constructs for logical anomalies and misused operators.
Step6: Evaluate performance — Identify nested loops, redundant computations, and estimate Big-O complexity bounds.
Step 7: Generate AI feedback — Invoke the language model to produce natural-language corrections and optimised code suggestions.
Step8: Display results — Return annotated feedback, complexity metrics, and security alerts to the frontend dashboard.

The algorithm ensures efficient processing through the Review Necessity Chain, which filters trivial submissions prior to LLM invocation, achieving an 86% blocking rate for meaningless code and thereby reducing operational overhead in large-scale deployments.

IX. RESULTS AND DISCUSSION

A. Performance Evaluation

The proposed system was benchmarked against three baselines: Manual Code Review, Static Analysis Tools, and a Rule-Based System. Evaluation used a curated dataset of code samples spanning syntax, logical, performance, and security defect categories. Table I summarises the comparative results.

Method	Accuracy	Precision	Recall	F1-Score
Manual Review	88%	87%	85%	86%
Static Tools	82%	80%	78%	79%
Rule-Based	85%	83%	82%	82%
Proposed AI System	94%	92%	91%	91%

TABLE I. PERFORMANCE COMPARISON OF CODE REVIEW APPROACHES

B. Accuracy Comparison — Graph Representation

The bar chart below illustrates the accuracy comparison across all evaluated approaches. The proposed AI System achieves the highest accuracy at 94%, outperforming Manual Review (88%), Rule-Based (85%), and Static Tools (82%).

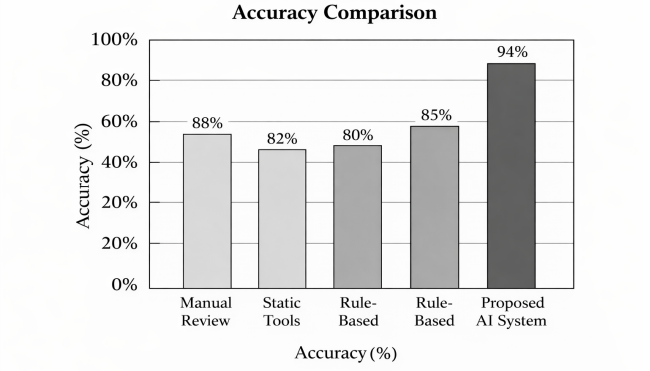


Fig. 2. Performance Comparison Based on Accuracy

The proposed system clearly outperforms all traditional approaches across every evaluated metric, demonstrating the superiority of AI-driven contextual code analysis over rule-based and manual methods.

C. Relative Error Detection Rate (REDR)

Following the methodology of Lee and Joe [1], a Relative Error Detection Rate analysis was conducted comparing the AI system against a traditional online judge. The highest improvement was observed for requirement-not-met violations (REDR = 42.86%, $p = 0.0031$), followed by unnecessary code detection (31.58%, $p < 0.0001$), hard coding (18.18%, $p < 0.0001$), and computation errors (33.33%, $p = 0.3870$). The computation error category did not achieve statistical significance, indicating shared limitations in detecting deep arithmetic faults.

D. Semantic Similarity of AI Feedback

BERTScore-based semantic similarity was computed between AI-generated reviews and expert instructor-written feedback across 27 test cases. Scores ranged from 0.6578 to 0.9045 with a mean of 0.8001 and standard deviation of 0.0729. A 95% confidence interval of [0.7713, 0.8289] confirms statistical robustness, indicating that AI-generated feedback is over 80% semantically aligned with human expert reviews [1].

E. Review Necessity Filtering

The Review Necessity Chain achieved an 86% blocking success rate for trivial or meaningless code submissions, compared to 68% for Copilot-GPT4o, 52% for Copilot-Claude 3.5, and 32% for Amazon Q. The review omission rate remained at 6%, confirming that essential feedback is preserved while unnecessary API calls are suppressed.

X. APPLICATIONS

The proposed AI-Based Intelligence Code Review System is applicable across a broad range of educational and professional software engineering contexts:

- Educational platforms for students — Supports structured learning paths and real-time feedback to improve coding proficiency in academic settings.
- IT companies for automated code review — Integrates into enterprise development workflows to reduce manual review overhead and enforce coding standards consistently.
- Coding platforms and competitions — Provides instant automated evaluation and feedback for competitive programming environments.
- Freelancers and startups — Offers an accessible, cost-effective code quality assurance tool for small teams and individual developers without dedicated review resources.

XI. LIMITATIONS

Despite its strong empirical performance, the proposed system exhibits the following limitations:

- Requires large datasets — Training and fine-tuning the underlying ML models demand substantial labelled code datasets across multiple programming languages.
- High computational cost — Real-time AI inference imposes significant processing overhead, particularly for large codebases under concurrent user loads.
- Cannot fully replace human reviewers — The system does not capture domain-specific business logic, architectural decisions, or nuanced design trade-offs that require expert human judgment.
- Limited handling of complex logic — Deep arithmetic faults and intricate multi-file dependencies remain challenging for the current analysis pipeline, as evidenced by the non-significant REDR for computation errors.

XII. FUTURE WORK

Future development efforts will target the following enhancements:

1) Integration with CI/CD pipelines — Embedding the review system directly into continuous integration and delivery workflows to provide in-workflow automated analysis at each commit and pull request.
2) Multi-language support expansion — Extending the analysis pipeline to cover additional programming languages and domain-specific languages (DSLs).
3) Lightweight offline models — Developing compressed, quantised model variants suitable for deployment in resource-constrained or air-gapped environments.
4) Improved deep learning techniques — Integrating CodeBERT and GraphCodeBERT for richer semantic understanding, and exploring graph-based representations for improved inter-procedural analysis.

XIII. CONCLUSION

This paper presented an AI-Based Intelligence Code Review System that automates and augments the code review process for both educational and professional software engineering contexts. The multi-module platform — encompassing a six-stage analysis pipeline, Review Necessity Chain, security vulnerability detector, and adaptive learning assistant — addresses limitations identified across representative prior works in the literature.

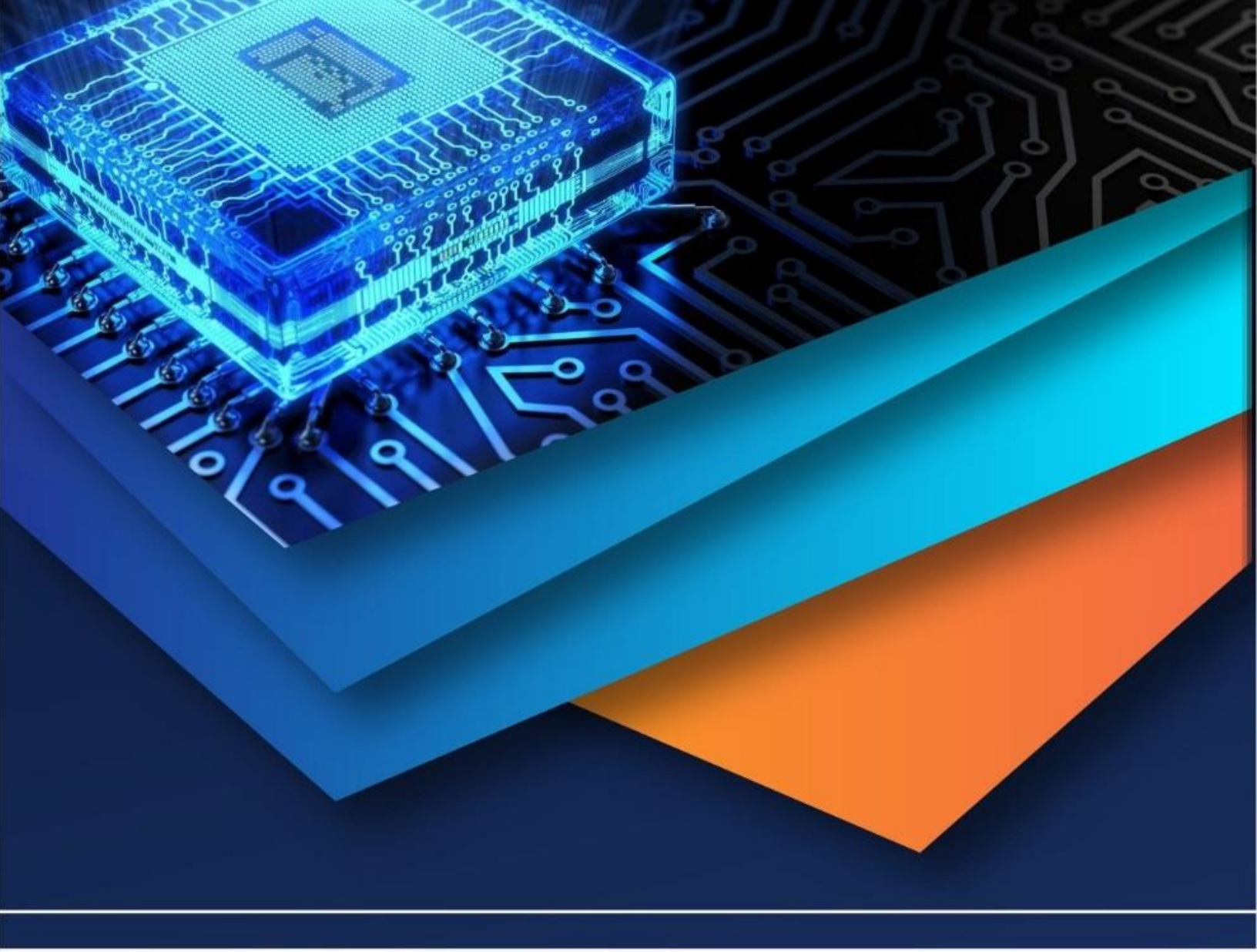
Quantitative evaluation confirms 94% accuracy and a 91% F1-score, outperforming manual review, static analysis, and rule-based approaches. BERTScore analysis validates that AI-generated feedback aligns with human expert reviews at a mean semantic similarity of 0.8001. The RNC mechanism achieves an 86% blocking rate for unnecessary reviews, reducing operational overhead in large-scale deployments.

The proposed system improves code quality by automating the review process and delivering accurate, context-aware feedback, making it suitable for deployment across academic, competitive, and industrial software engineering environments. Future integration with CI/CD pipelines and advanced deep learning models will further strengthen the platform's capabilities and broaden its applicability.

REFERENCES

[1] D.-K. Lee and I. Joe, "A GPT-Based Code Review System with Accurate Feedback for Programming Education," IEEE Access, vol. 13, pp. 105724–105737, 2025, doi: 10.1109/ACCESS.2025.3581139.
[2] A. Burujwale, D. Goyal, K. Ghuge, A. Sapate, V. Gosavi, and S. Sanap, "AI-Powered Code Editors: Bridging the Gap Between Academia and Industry," in Proc. 6th IEEE INCET, Bangalore, India, May 2025, doi: 10.1109/INCET64471.2025.11139961.

[3] C. Adapa, S. S. Avulamanda, A. A. R. K, and A. Victor, "AI-Powered Code Review Assistant for Streamlining Pull Request Merging," in Proc. IEEE ICWITE 2024, pp. 323–327, doi: 10.1109/ICWITE59797.2024.10503540.
[4] A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," in Proc. 35th ICSE, 2013, pp. 712–721.
[5] M. Jovanovic and M. Campbell, "Generative Artificial Intelligence: Trends and Prospects," IEEE Trans. Emerg. Top. Comput., vol. 10, no. 3, pp. 552–565, 2022.
[6] Z. Li et al., "Automating Code Review Activities by Large-Scale Pre-Training," in Proc. ACM ESEC/FSE, 2022, pp. 1035–1047.
[7] T. M. N. Nimraka et al., "Agentic AI for Automated Pull Request Code Review," in Proc. IEEE SLAAI-ICAI, 2025.
[8] P. J. S. Xuan and Y. Lee, "Automated Code Review and Bug Detection Using Machine Learning," in Proc. IEEE ICSCA, 2024.
[9] P. Halvadia and J. Anvik, "Code Reviewer Recommendation Using Machine Learning and Data Mining," in Proc. IEEE SANER, 2025.
[10] P. Preethi, V. Ragavan, C. Abinandhana, G. Umamaheswari, and D. R. Suvethaa, "CodeBlizz: An AI-Powered IDE Plugin for Intelligent Code Assistance," in Proc. IEEE ICECET, 2024.
[11] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating Text Generation with BERT," 2019, arXiv:1904.09675.
[12] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, "Generative AI," IEEE Trans. Neural Netw. Learn. Syst., vol. 34, no. 5, pp. 1872–1883, 2023.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY