IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510139

AI-Powered Automated Code Review for GitHub: A Survey of Static Analysis, Large Language Models, Hybrid Approaches, and Developer Workflows

Archana Raj[1], Pavithra J[2], Hima Shaji[3], Jyolsna K V[4]

Department of Computer Science & Engineering, College of Engineering Kottarakkara, Kollam, Kerala, India [1-4]

Abstract: Automated code review has become an important research direction in software engineering because conventional manual review is time-consuming and dependent on reviewer expertise. Traditional static analysis tools can efficiently identify predefined coding violations and security weaknesses, but they have limited understanding of program intent and may produce false-positive warnings. Recent advances in Large Language Models (LLMs) provide new capabilities for code understanding, defect detection, explanation generation, and automated review. This survey examines recent research on AI-powered automated code review, covering static analysis, machine learning, LLM-based approaches, hybrid analysis, automated program repair, vulnerability detection, and GitHub-based workflows. Twenty research papers are comparatively analyzed with respect to methodology, tools, datasets, evaluation metrics, advantages, limitations, and applicability to automated GitHub code review. The survey identifies important research gaps, including false positives, hallucination, limited contextual understanding, computational cost, language-specific analysis, and limited integration with developer workflows. Based on these findings, the survey motivates a hybrid architecture that combines deterministic static analysis with context-aware LLM reasoning and GitHub Pull Request integration.

Keywords: Automated Code Review, Large Language Models, Static Analysis, GitHub, Software Engineering, Code Intelligence, Code Security

I. INTRODUCTION

Code review is an essential software engineering activity for identifying defects, improving code quality, enforcing coding standards, and detecting security issues before software changes are integrated. With modern collaborative development, GitHub Pull Requests have become an important part of the review process. However, manually reviewing a large number of code changes can be time-consuming and difficult to scale.

Traditional automated code-review techniques mainly rely on static analysis, linters, and rule-based tools. These methods are efficient and deterministic for detecting predefined coding violations and security patterns, but they have limited ability to understand developer intent and broader program context. They may also produce false-positive warnings or fail to identify issues that require contextual reasoning.

Recent advances in Large Language Models (LLMs) have introduced new possibilities for automated code review. LLMs can analyze source code, interpret code changes, generate explanations, and suggest possible improvements. However, LLM-based systems may produce inaccurate or irrelevant feedback and can introduce additional computational cost. These limitations motivate hybrid approaches that combine the reliability of static analysis with the contextual reasoning capabilities of LLMs.

This survey reviews research on AI-powered automated code review, including static analysis, machine learning, LLM-based approaches, hybrid systems, automated program repair, vulnerability detection, and developer-oriented workflows. Twenty research papers are comparatively analyzed based on their techniques, contributions, datasets, evaluation approaches, and limitations. The findings are used to identify research gaps and motivate the development of an AI-powered GitHub code-review system that combines static analysis with LLM-based reasoning to provide relevant and explainable feedback to developers.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510139

A. Background and Motivation

The increasing volume of software changes makes automated assistance valuable for modern code-review workflows. Static analysis provides fast and reproducible findings, whereas LLMs can interpret code and generate natural-language explanations. Combining these capabilities can provide a more useful review process while retaining developers as the final decision-makers. The proposed project focuses on integrating automated code analysis with GitHub Pull Requests. The intended workflow analyzes changed files using appropriate static-analysis tools, aggregates the detected findings, and uses an LLM to provide contextual interpretation, severity information, and explanations. The resulting feedback can then be presented to developers through the GitHub review workflow.

B. Contributions of This Survey

This survey provides a structured analysis of twenty research papers related to automated code review and AI-assisted software engineering. It classifies existing approaches, compares their techniques and contributions, examines their limitations, and identifies research gaps relevant to reliable, contextual, and GitHub-integrated automated code review.

C. Paper Organization

The remainder of this paper is organized as follows. Section II describes the survey methodology. Section III presents the taxonomy of AI-powered code-review approaches. Section IV reviews the selected research works. Section V discusses datasets and evaluation benchmarks. Section VI presents the comparative analysis. Section VII discusses research gaps and project motivation. Section VIII describes challenges and future directions. Section IX concludes the survey.

II. METHODOLOGY OF SURVEY

This survey follows a structured approach to identify and analyze research related to AI-powered automated code review. The selected studies cover static analysis, machine learning, Large Language Models (LLMs), hybrid analysis, automated program repair, vulnerability detection, and code-review automation. The reviewed papers are compared based on their techniques, contributions, evaluation approaches, and research limitations.

A. Data Sources

The research papers considered in this survey were collected from academic research sources, with particular emphasis on IEEE Xplore and related research repositories. The selected literature focuses on automated code analysis, software engineering, artificial intelligence, LLM-based code review, static analysis, and developer workflows.

B. Search Keywords

Relevant studies were identified using keywords such as “automated code review,” “AI code review,” “LLM code review,” “large language models for software engineering,” “static analysis and LLM,” “GitHub Pull Request automation,” “automated program repair,” and “LLM vulnerability detection.” These keywords were used to identify research addressing different aspects of automated code analysis and review.

C. Inclusion Criteria

Studies were included when they presented a method, model, tool, or framework related to automated code analysis or code review. Papers addressing static analysis, machine learning, LLM-based code understanding, vulnerability detection, automated program repair, code-review automation, or software-engineering applications of LLMs were considered relevant to the survey.

D. Exclusion Criteria

Studies that were not related to automated code analysis, code review, software engineering, program repair, vulnerability detection, or AI-assisted development were excluded. Research that did not provide sufficient information about its proposed approach or evaluation was also considered unsuitable for the comparative analysis.

E. Papers Reviewed

A total of twenty research papers were selected for detailed analysis. The papers represent different approaches to automated code review, ranging from traditional static analysis to modern LLM-based and hybrid techniques. Each study was examined with respect to its methodology, main contribution, evaluation approach, and limitations in order to identify common trends and research gaps relevant to the proposed AI-powered GitHub code-review system.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510139

III. TAXONOMY OF AI-POWERED CODE REVIEW

AI-powered automated code review can be classified into several major approaches based on the techniques used for analyzing source code and generating review feedback. The main categories identified in the reviewed literature are static analysis, machine learning and deep learning, Large Language Model (LLM)-based review, hybrid static analysis–LLM approaches, automated program repair, and human-in-the-loop code review. The taxonomy is illustrated in Fig. 1.

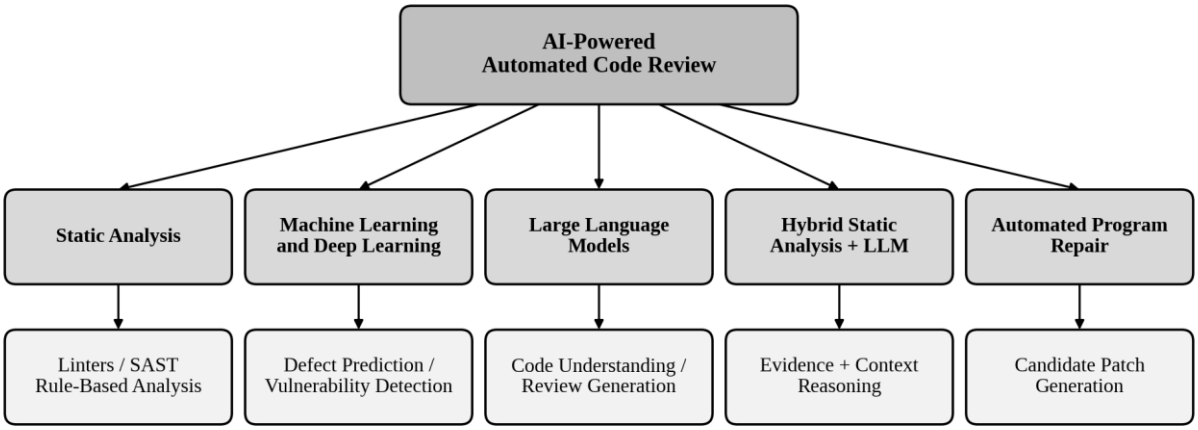


Fig. 1. Taxonomy of AI-powered automated code review approaches.

A. Static Analysis-Based Approaches

Static analysis examines source code without executing the program. Traditional linters, rule-based analyzers, Abstract Syntax Tree (AST) analysis, and Static Application Security Testing (SAST) tools are widely used to detect coding violations, security vulnerabilities, and predefined defect patterns. These approaches are generally fast, deterministic, and suitable for continuous integration environments. However, they may generate false positives and have limited ability to understand developer intent or complex program context.

B. Machine Learning and Deep Learning Approaches

Machine learning and deep learning techniques learn patterns from existing source code, defect datasets, and software engineering artifacts. These approaches have been applied to defect prediction, code classification, vulnerability detection, and code quality assessment. Unlike rule-based systems, learning-based methods can identify patterns from data, but their performance depends on the quality, size, and representativeness of the training datasets.

C. Large Language Model-Based Approaches

Large Language Models use transformer-based architectures to understand and generate programming-language text. In automated code review, LLMs can analyze code changes, identify potential problems, explain detected issues, and generate improvement suggestions. Their ability to process natural language and source code makes them useful for contextual review. However, LLMs may produce incorrect or hallucinated findings and therefore require validation.

D. Hybrid Static Analysis–LLM Approaches

Hybrid approaches combine conventional static analysis with the contextual reasoning capabilities of LLMs. Static analyzers first identify deterministic findings, which are then supplied to an LLM along with relevant source-code context. The LLM can prioritize findings, remove irrelevant warnings, generate explanations, and suggest possible corrections. This approach is particularly suitable for the proposed AI-powered GitHub code-review system.

E. Automated Program Repair

Automated program repair focuses on generating modifications to correct detected defects. Traditional repair systems use predefined repair strategies, while recent LLM-based approaches can generate candidate patches from source-code context. Generated patches must be validated through testing or additional static analysis before they are accepted.

F. Human-in-the-Loop Code Review

Human-in-the-loop systems retain developers as the final decision-makers. The automated system provides detected issues, severity information, explanations, and possible fixes, while the developer can accept, reject, or modify the

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510139

recommendations. This approach improves reliability and developer trust by preventing unverified AI suggestions from being automatically applied.

IV. LITERATURE REVIEW

A. Static Analysis for Code Review

Static analysis remains an important foundation for automated code review because it provides deterministic findings. Traditional linters and SAST tools are particularly useful for coding standards, security rules, and known vulnerability patterns. However, their rule-based nature can result in large numbers of warnings and may not capture deeper logical errors.

B. Machine Learning for Code Analysis

Machine learning methods have been investigated for defect prediction, code classification, vulnerability detection, and other software-engineering tasks. These approaches can learn patterns from historical code and defect datasets, although their performance depends strongly on the quality and representativeness of training data.

C. Large Language Models for Code Review

Recent research demonstrates the potential of LLMs for automated code review. LLMs can analyze code changes and generate natural language explanations. Industrial studies have also examined their use in real development environments and Pull Request workflows.

D. Hybrid Static Analysis and LLM Systems

Hybrid systems attempt to address weaknesses of both approaches. Static analyzers provide reliable evidence about known issues, while LLMs can interpret that evidence using the surrounding code context. This combination is particularly relevant to the proposed GitHub-based project.

E. Security and Vulnerability Detection

Security analysis is an important component of automated code review. Static security analyzers can detect known vulnerability patterns, while AI-based methods may identify more complex security issues. Combining both approaches can improve the breadth of review.

F. Automated Program Repair

Automated program repair focuses on generating modifications that correct detected defects. LLM-based repair can produce readable candidate patches, although generated changes must be validated before being accepted.

G. GitHub and Developer Workflow Integration

GitHub Pull Requests provide a practical integration point for automated code review. A review system can detect a new Pull Request, analyze changed files, generate findings, and post structured feedback directly to the Pull Request.

H. Human-in-the-Loop and Explainable Review

Developer trust is important for automated review. Systems should provide understandable explanations and allow developers to verify or reject recommendations instead of treating AI output as automatically correct.

V. DATASETS AND EVALUATION BENCHMARKS

A. Code Review Datasets

Code review datasets generally contain source-code changes, Pull Requests, review comments, and developer actions. Such data can be used to evaluate the quality and usefulness of automated review systems.

B. Bug and Defect Datasets

Bug datasets contain defective and corrected code examples. They are commonly used to evaluate defect detection and automated repair systems.

C. Vulnerability Datasets

Security datasets contain vulnerable and secure code examples and are useful for evaluating vulnerability-detection approaches.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510139

D. GitHub Pull Request Data

GitHub provides real-world Pull Requests and review discussions that can be used to evaluate automated review in realistic development settings.

E. Evaluation Metrics

Common evaluation measures include precision, recall, F1-score, accuracy, false-positive rate, review latency, and developer acceptance or resolution rate.

VI. COMPARATIVE ANALYSIS

The reviewed literature demonstrates the evolution of automated code review from rule-based static analysis and traditional machine learning toward transformer-based and Large Language Model (LLM) approaches. The studies differ in their objectives, analysis techniques, datasets, and degree of integration with real-world software-development workflows. Table I presents a comparative analysis of research works from the reviewed literature based on their authors, techniques, main contributions, and limitations.

A. Comparison of Reviewed Research Papers

TABLE I COMPARATIVE ANALYSIS OF RESEARCH PAPERS

Author / Paper	Technique	Main Contribution	Limitation
Gupta and Sundaresan	LSTM-based deep learning	Uses positive and negative code–review pairs to retrieve relevant review comments and predict their relevance for code changes.	Depends on historical review data and has limited ability to reason about complex repository-level context.
Siow et al.	Multi-level embeddings	Automates review recommendation by using word-level and character-level representations to capture relationships between code changes and review comments.	Primarily focuses on review recommendation rather than complete automated review and contextual defect reasoning.
Tufano et al.	Transformer-based code review	Investigates transformer models for automating code-review activities and generating useful code-review-related outputs.	Performance depends on training data and generated results may not fully capture the context of real software projects.
Li et al.	CodeT5 / large-scale pre-training	Introduces a large-scale pre-trained model for automating code review activities including review comment generation and related code-review tasks.	Requires substantial training data and computational resources; generalization across different projects can remain challenging.
Sghaier and Sahraoui	Cross-task knowledge distillation	Improves learning of successive code-review tasks by transferring knowledge between related tasks and exploiting their interdependence.	The approach depends on multiple related review tasks and does not directly provide a complete GitHub Pull Request review pipeline.
Pornprasit and Tantithamthavorn	LLM fine-tuning and prompt engineering	Systematically compares GPT-3.5 and Magicoder using fine-tuning, zero-shot, few-shot, and persona-based prompting for code-review automation.	Fine-tuning can be expensive; prompting performance varies with available examples and prompt configuration.
Tufano et al.	Pre-trained transformer models	Uses pre-trained models to improve automated code-review tasks and demonstrates the benefit of transferring knowledge from large pre-trained models.	Primarily evaluated on specific datasets and tasks, limiting direct application to multi-language GitHub Pull Request workflows.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510139

Author / Paper	Technique	Main Contribution	Limitation
Li et al.	Pre-trained models	Auger automatically generates code-review comments using pre-trained models to improve automated review generation.	Generated comments can suffer from incorrect or incomplete information and require reliable validation.
Lu et al.	Parameter-efficient LLM fine-tuning	Llama-Reviewer applies parameter-efficient fine-tuning to large language models for automated code review.	Fine-tuning remains dependent on suitable review datasets and model quality; hallucinated or irrelevant comments may still occur.
Wadhwa et al.	LLM-based code-quality analysis	Uses LLMs to resolve code-quality issues and provides contextual reasoning beyond conventional rule-based analysis.	LLM-based reasoning can introduce computational cost and non-deterministic results compared with traditional static analysis.

VII. RESEARCH GAPS AND PROJECT MOTIVATION

A. Limitations of Existing Static Analysis

Static analyzers are effective for predefined rules but can produce false-positive warnings and have limited ability to understand developer intent or broader application context.

B. Limitations of LLM-Based Code Review

LLM-based systems can generate useful explanations but may produce incorrect, irrelevant, or hallucinated review comments. Their outputs therefore require validation.

C. False Positives and Hallucinations

A practical automated review system must reduce unnecessary warnings while maintaining sensitivity to genuine defects. Combining static evidence with LLM reasoning provides one possible mechanism for improving review relevance.

D. Multi-Language Support

Many existing systems focus on a particular programming language or limited set of analysis tools. Supporting multiple languages requires appropriate language-specific static analyzers and context-aware processing.

E. GitHub-Centered Automation

Some research focuses primarily on code analysis without providing a complete Pull Request workflow. Direct integration with GitHub can make automated review more practical for real development teams.

F. Context-Aware Code Review

Review quality can depend on repository context, surrounding code, project conventions, and the purpose of a change. Future systems should provide relevant repository context to the analysis model.

G. Human Feedback and Explainability

Developers need understandable and actionable feedback. Systems should support human verification and learn from accepted or rejected recommendations.

H. Security and Privacy

Source code may contain proprietary information. Secure processing, access control, data protection, and appropriate handling of external LLM services are therefore important considerations.

VIII. CHALLENGES AND FUTURE DIRECTIONS

A. LLM Reliability

Future systems should improve factual reliability and reduce hallucinated explanations or incorrect fixes.

B. False Positives and False Negatives

Balancing detection sensitivity with useful feedback remains a major challenge. Hybrid verification and severity ranking may reduce unnecessary comments.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510139

C. Computational Cost

Large language models can require significant computational resources. Efficient models and selective invocation can reduce latency and cost.

D. Security and Privacy

Future implementations should consider secure deployment, controlled access to repositories, and protection of source-code information.

E. Multi-Language Code Analysis

Supporting languages such as Python, JavaScript, Java, and other commonly used programming languages can improve the applicability of automated review systems.

F. GitHub and CI/CD Integration

Automated review should operate naturally within existing Pull Request and CI/CD workflows without introducing unnecessary delays.

G. Explainable AI and Developer Trust

Review comments should clearly explain the detected problem, severity, evidence, and possible correction so that developers can make informed decisions.

H. Future Research Opportunities

Promising directions include Retrieval-Augmented Generation (RAG), repository-level context, feedback-driven learning, multi-agent review systems, automated patch validation, and improved human-in-the-loop workflows.

IX. CONCLUSION

This survey reviewed recent research on AI-powered automated code review, covering static analysis, machine learning, Large Language Models, hybrid analysis, automated program repair, vulnerability detection, and GitHub-based workflows. The literature indicates that static analysis provides fast and deterministic findings, whereas LLMs provide stronger contextual reasoning and natural language explanations. However, both approaches have limitations when used independently. Hybrid approaches therefore provide a promising direction for practical automated code review. The identified research gaps motivate the development of a GitHub-based system that combines static analysis with LLM reasoning to provide context-aware, prioritized, and actionable feedback to developers.

REFERENCES

[1]. "AI-powered code reviews: Leveraging large language models," in *2024 International Conference on Signal Processing and Advance Research in Computing (SPARC)*, 2024.
[2]. "Automated code review in practice," in *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025.
[3]. "On the performance of large language models for code change intent classification," 2025.
[4]. "Automating patch set generation from code review comments using large language models," 2024.
[5]. "Exploring parameter-efficient fine-tuning of large language model on automated program repair," in *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024.
[6]. "What do pre-trained code models know about code?" in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021.
[7]. "CodeBERT for code clone detection: A replication study," 2022.
[8]. "Assessing generalizability of CodeBERT," 2021.
[9]. "Python source code vulnerability detection based on CodeBERT language model," 2025.
[10]. "DP-CCL: A supervised contrastive learning approach using CodeBERT model in software defect prediction," 2024.
[11]. "Greening large language models of code," in *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, 2024.
[12]. "LLMLoop: Improving LLM-generated code and tests through automated iterative feedback loops," in *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2025.
[13]. "SALLM: Security assessment of generated code," in *2024 39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, 2024.



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510139

[14]. "Follow-up attention: An empirical study of developer and neural model code exploration," *IEEE Transactions on Software Engineering*, vol. 50, no. 10, pp. 2568–2582, 2024.
[15]. "On the effectiveness of large language models in statement-level code summarization," in *2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS)*, 2024.
[16]. "Software service engineering in the era of large language models," in *2024 IEEE International Conference on Software Services Engineering (SSE)*, 2024.
[17]. "Machine/deep learning for software engineering: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 1188–1231, 2023.
[18]. "AI-driven continuous integration: Automating code review and deployment with LLMs," in *2025 10th International Conference on Fog and Mobile Edge Computing (FMEC)*, 2025.
[19]. "Understanding the robustness of transformer-based code intelligence via code transformation: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 51, no. 2, pp. 521–547, 2025.
[20]. "Application of large language models to software engineering tasks: Opportunities, risks, and implications," 2023.