RESEARCH ARTICLE

DOI: 10.34256/ijcci2614

International Journal of Computer Communication and Informatics

AI-Powered Intelligent Code Review and Pull Request Analysis System

T. Manojpraphakar [a], K.E. Kannammal [b], P. Sasikala [b], S. Kalandhar Naina Mohamed [b]

[a] Department of Computer Science and Engineering, Sri Ramakrishna Engineering College, Coimbatore, India

[b] Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, India

* Corresponding Author: manojpraphakar@srec.ac.in

Received: 27-01-2026, Revised: 13-04-2026, Accepted: 22-04-2026, Published: 03-05-2026

Abstract: Modern code review is essential for defect prevention, maintainability, knowledge transfer, and collaborative software development, but manual review is frequently delayed by reviewer workload and inconsistent review depth. This paper presents ReviewPilot, a web-based prototype that integrates repository import, pull request analysis, static checks, large-language-model-assisted review, configurable review rules, issue summarisation, and a composite Code Health Score within a unified dashboard. The proposed workflow ingests repository and pull request metadata, validates and parses changed files, applies deterministic rules and conventional static analysis, and then invokes an AI review layer to generate context-sensitive observations. The results are organised by severity, file, and review category before being displayed through repository, pull request, rules, review-history, and dashboard interfaces. Functional interface testing confirms that the principal modules and navigation flow operate as intended in the simulated environment. However, the present work is a proof-of-concept system demonstration rather than a comparative evaluation of defect-detection accuracy, review usefulness, latency, security, or developer productivity. Accordingly, the paper defines an evaluation framework for future studies using labelled code changes, expert reviewer judgements, precision, recall, false-positive rate, response time, and user-centred measures. ReviewPilot therefore demonstrates a technically coherent architecture for AI-assisted review while recognising that human oversight, reproducible benchmarking, privacy controls, and hallucination mitigation remain necessary before production deployment.

Keywords: Automated Code Review, Pull Request Analysis, Large Language Models, Static Analysis, Software Quality, Code Health, Developer Tools.

1. Introduction

Modern code review is a lightweight quality-assurance practice in which developers examine proposed changes before integration. Beyond defect identification, review supports knowledge transfer, team awareness, maintainability, and shared ownership of a codebase [1]. Empirical studies have also associated review coverage and reviewer participation with post-release software quality [2]. Large-scale industrial practice demonstrates that effective review depends not only on tooling, but also on small changes, timely feedback, clear ownership, and reviewable context [3]. Nevertheless, manual review remains cognitively demanding and can become a delivery bottleneck when pull request volume, codebase size, or reviewer workload increases [4].

Recent research has therefore shifted from rule-based linting toward learning-based review assistance. Pre-trained models specialised for code changes can estimate change quality, generate review comments, and recommend code refinements [5], while transfer-learning approaches have improved the automation of code-review tasks under more realistic conditions [6]. Large-language-model-based systems further extend this capability by interpreting code context and producing natural-language explanations or recommendations [7], although their performance depends strongly on prompting strategy, task context, and domain adaptation [8].

Despite this progress, automated comments may be incomplete, irrelevant, overly general, or factually unsupported. Recent evaluations emphasise the need for human-grounded quality metrics [9], and empirical evidence shows that generated review comments can contain hallucinated claims [10]. Industrial studies likewise report mixed outcomes: AI-assisted review can increase awareness of quality concerns, yet may also introduce unnecessary corrections and extend pull request closure time [11]. These findings indicate that AI review should complement rather than replace deterministic analysis and expert judgement.

Against this background, this paper presents ReviewPilot, a web-based proof-of-concept platform that combines repository import, pull request inspection, static analysis, configurable review rules, AI-generated review assistance, issue visualisation, and a Code Health Score. The contribution is primarily architectural and functional: the work defines an integrated workflow, demonstrates the principal user interfaces, and identifies the validation requirements needed before the platform can be treated as an empirically established software-engineering tool.

2. Related Work

Foundational studies of modern code review show that review outcomes extend beyond bug finding to include learning, communication, and alternative solution development [1]. Subsequent empirical work demonstrated that review coverage and participation can influence software quality, reinforcing the importance of consistent and traceable review processes [2].

Google's industrial case study further illustrated how integrated tooling and well-defined review practices can support review at organisational scale [3].

Research on automation initially focused on static analysers, defect prediction, and recommendation systems. More recent work has applied pre-trained code models to review-specific tasks. CodeReviewer introduced large-scale pre-training tailored to code changes and review artefacts [5], while T5-based transfer learning improved code-review automation on larger and more realistic datasets [6]. LLaMA-Reviewer and related LLM-based systems investigated natural-language review generation and contextual code assessment [7].

Prompt engineering and fine-tuning studies indicate that the quality of automated review depends on the information supplied with a code change, including semantic summaries and structural context [8]. Evaluation remains a major challenge because conventional reference-overlap metrics do not reliably represent whether a comment is correct, specific, actionable, and grounded in the changed code. CRScore addresses this limitation through human-annotated, claim-grounded assessment of review quality [9].

The reliability of generated feedback is equally important. Analyses of code-change-to-text generation have documented hallucinated review statements, demonstrating that fluent comments are not necessarily technically valid [10]. Industrial evidence also suggests that AI review is useful only when developers can verify, dismiss, or act on each recommendation and when the tool is integrated without disrupting established pull request workflows [11]. Experience-aware training and quality-oriented data selection may further improve the correctness and informativeness of generated reviews [12].

ReviewPilot is positioned within this emerging hybrid paradigm. Rather than presenting the language model as an autonomous reviewer, the platform combines deterministic checks, repository context, configurable rules, AI-assisted interpretation, severity grouping, and human decision-making. This architecture is intended to improve traceability and reduce dependence on any single analysis mechanism.

3. Limitations of Existing Approaches

Conventional review workflows typically combine manual inspection with platform-specific pull request interfaces, linters, test reports, and security scanners. Although these mechanisms are valuable, they are often fragmented across multiple tools and may not provide a consolidated explanation of maintainability, risk, and review status. Manual review quality can also vary with reviewer expertise, familiarity with the codebase, available time, and change complexity. Conversely, purely generative systems may produce plausible but unsupported comments, overlook project-specific constraints, or expose sensitive source code when external services are used. A practical review platform should therefore integrate deterministic checks,

contextual AI assistance, explicit rule configuration, provenance-aware outputs, and human approval rather than relying on an opaque automated score.

Principal limitations of existing approaches include:

- Fragmented analysis across separate repository, static-analysis, security, and review tools.
- Inconsistent review depth caused by workload, expertise, and codebase familiarity.
- Limited contextual understanding in conventional rule-based analysers.
- False positives, hallucinated explanations, privacy risks, and weak reproducibility in unconstrained AI review.

4. Proposed System

ReviewPilot is proposed as a web-based, human-in-the-loop code-review support platform. A user authenticates, connects or imports a repository, selects a branch or pull request, and initiates analysis. The backend retrieves the relevant diff and repository metadata, applies syntax and rule validation, executes available static checks, and prepares a bounded context for the AI review service. Generated observations are mapped to files and changed lines, grouped by category and severity, and presented with an explanation and suggested action. Repository-specific rules allow teams to define review priorities, excluded paths, coding conventions, and security or maintainability concerns. The Code Health Score is presented as a configurable summary indicator derived from measurable subcomponents rather than as an independently validated quality metric. Final acceptance, rejection, or modification of every recommendation remains with the developer or reviewer.

Key functional advantages

- Unified repository import, pull request tracking, issue presentation, and review history.
- Hybrid analysis combining deterministic rules, static checks, and AI-assisted interpretation.
- Configurable project-specific review policies and path-level focus rules.
- Traceable issue grouping by file, severity, and review category.
- Dashboard-based visibility of review status and repository-level trends.
- Authenticated access and a design that can be extended with role-based permissions and audit logs.

- ❖ Scalable modular architecture suitable for additional languages, analysers, and CI/CD integrations.

5. Methodology

The prototype follows a modular analysis pipeline. First, the user submits repository details, a pull request identifier, branch information, or selected source files through the web interface. The ingestion layer validates the request, retrieves authorised repository content, and constructs the change set. Second, a preprocessing layer identifies file types, removes unsupported binary artefacts, segments large diffs, and attaches contextual metadata such as file paths, changed functions, and repository rules. Third, deterministic analysers perform syntax validation, linting, rule matching, and available complexity or security checks. Fourth, the AI review layer receives only the bounded code context required for the selected task and generates candidate observations.

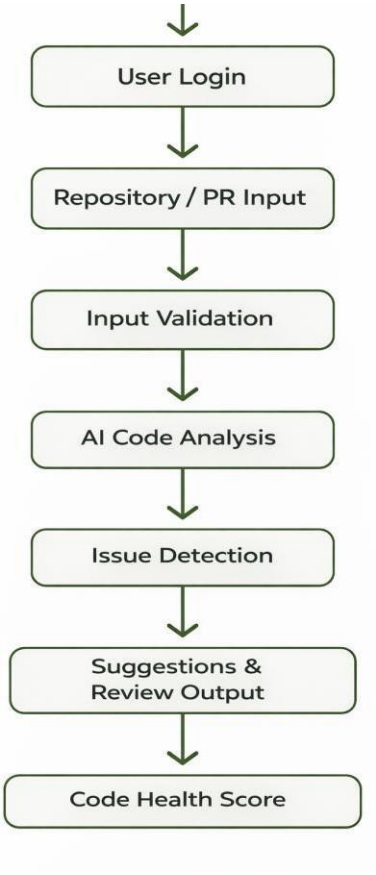

Figure 1. Proposed workflow for repository ingestion, deterministic analysis, AI-assisted review, result consolidation, and human decision-making.

Fifth, a post-processing layer removes duplicates, checks whether comments are anchored to the submitted change, assigns severity and category labels, and records provenance. Finally, the interface displays review findings, the calculated health indicators, and the reviewer's disposition of each item.

The workflow shown in Figure 1 separates deterministic analysis from generative interpretation and preserves a human approval stage. This distinction is necessary because existing studies show that LLM-generated review text must be evaluated for correctness, relevance, and grounding [9, 10]. For production use, credentials should be encrypted, access should follow least-privilege principles, sensitive source code should not be retained beyond the configured period, and every model request and response should be auditable. A rigorous evaluation should compare ReviewPilot with human review and established analysers using labelled pull requests, precision, recall, F1-score, false-positive rate, comment usefulness, response latency, cost, and developer acceptance.

The end-to-end ReviewPilot analysis and decision workflow is illustrated in Figure 1.

4. Prototype Results and Interface Validation

4.1 Landing-Page Validation

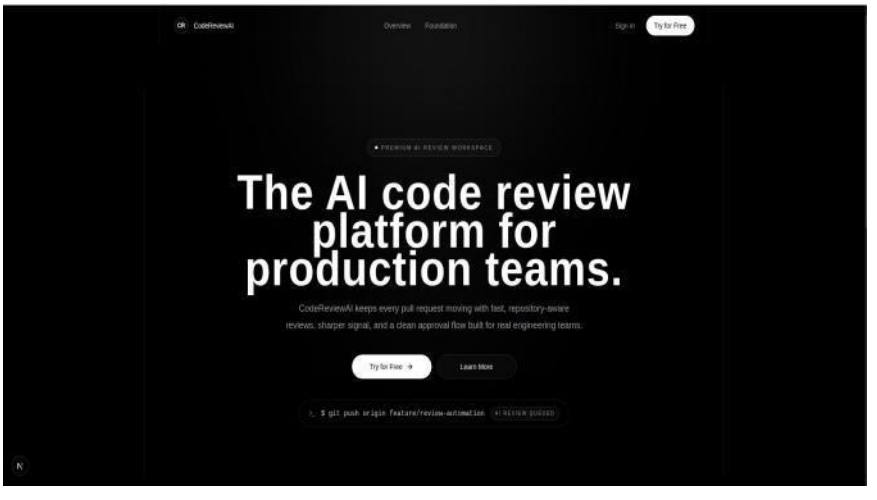

Figure 2. Review Pilot landing page and primary navigation interface.

Figure 2 shows the landing page used to introduce the platform and provide access to its principal functions. Functional inspection confirmed that the main navigation elements and calls to action are visible and logically organised; this observation concerns interface operation only and does not establish usability or productivity improvement.

4.2 Workflow-Information Page

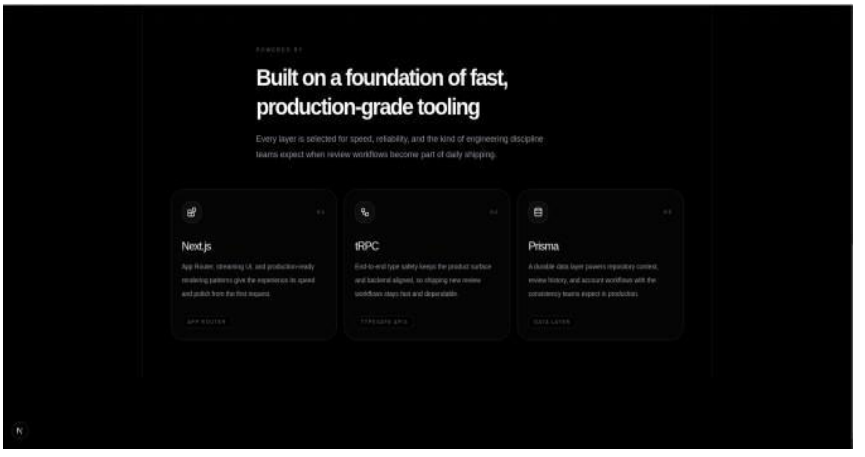

Figure 3. Workflow-information page describing the principal stages of Review Pilot.

As presented in Figure 3, the workflow-information page explains account creation, repository connection, pull request submission, automated analysis, and review presentation. The page provides procedural guidance but requires formal usability testing to determine whether first-time users can complete these tasks efficiently.

4.3 Authentication Interface

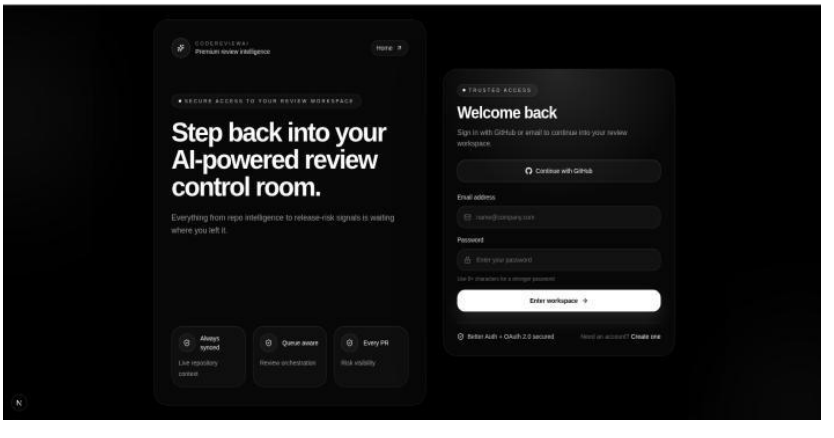

Figure 4. Sign-up and login interface for authenticated platform access.

Figure 4 presents the authentication interface through which users create an account or sign in. The prototype demonstrates the expected access flow; however, production readiness would require password-policy enforcement, secure session management, account recovery, rate limiting, and independent security testing.

4.4 Dashboard Interface

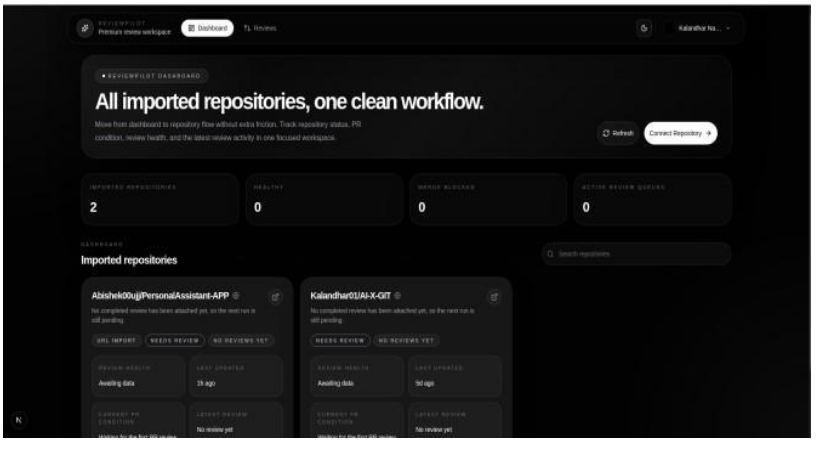

Figure 5. Dashboard summarising repositories, pull requests, findings, and code-health indicators.

The central dashboard shown in Figure 5 consolidates imported repositories, pull request status, detected findings, review history, and summary indicators. The displayed Code Health Score should be interpreted as a prototype aggregation of configured metrics; its construct validity and weighting have not yet been established through empirical study.

4.5 Repository-Import Interface

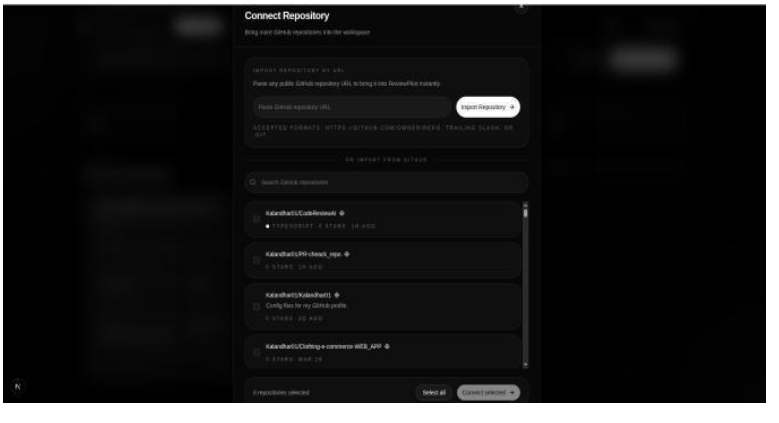

Figure 6. Repository-import interface for supplying repository, branch, and pull request information.

Figure 6 illustrates the repository-import interface. The module supports entry of repository and branch information and initiates the analysis workflow. Future implementation

should include explicit permission scopes, token revocation, repository-size limits, unsupported-file handling, and transparent error messages.

4.6 Repository Review Workspace

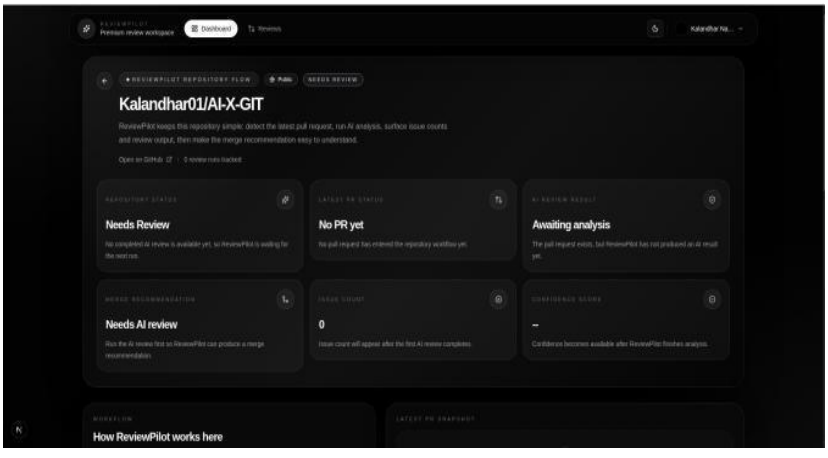

Figure 7. Repository review workspace displaying review status and analysis outputs.

The repository review workspace in Figure 7 provides a unified view of pull request progress, generated findings, and merge-related information. The screen demonstrates functional integration among interface modules, but the correctness of the findings must be tested against expert-labelled code changes before claims of review accuracy can be made.

4.7 Pull Request Review Workflow

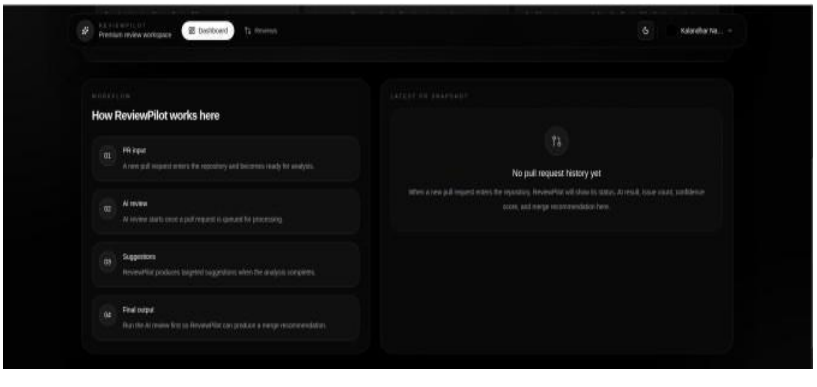

Figure 8. Pull request review workflow from submission to reviewer disposition.

As shown in Figure 8, the pull request workflow proceeds from input and contextual analysis to candidate review comments and reviewer action. This staged presentation improves

traceability because users can distinguish submitted changes, generated observations, and final decisions.

4.8 Review-Rule Configuration

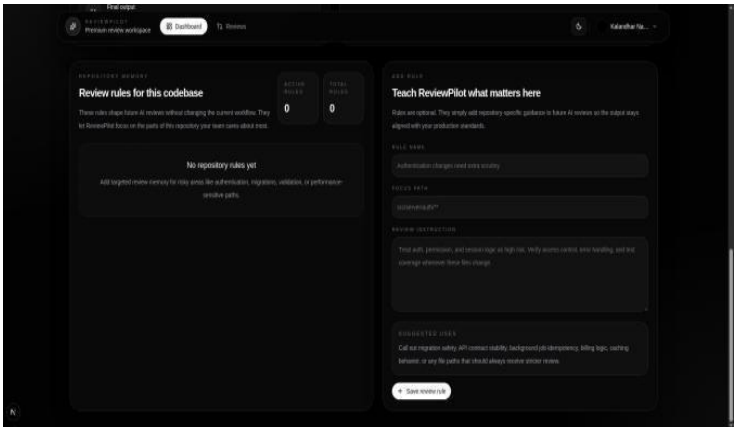

Figure 9. Rule-configuration interface for project-specific review instructions.

Figure 9 presents the rule-configuration interface, which allows users to specify rule names, target paths, and review instructions. Project-specific constraints can improve contextual relevance, but contradictory, overly broad, or insecure instructions should be detected and rejected by validation controls.

4.9 Review-History and Monitoring Interface

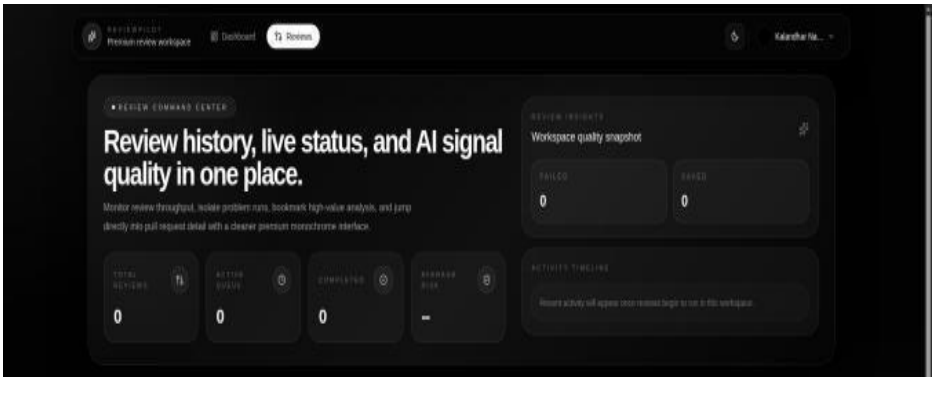

Figure 10. Review-history interface showing completed and active review activity.

The review-history interface in Figure 10 consolidates active and completed analyses, risk labels, quality indicators, and recent activity. This page can support auditability and trend

monitoring once the underlying measures, event timestamps, and retention rules are formally defined.

Overall, the prototype demonstrates a coherent navigation structure and the functional presence of the intended modules. No quantitative experiment, benchmark dataset, comparative baseline, sample size, statistical analysis, or developer study was provided in the source manuscript. Consequently, the results should be interpreted as interface and workflow validation rather than evidence of improved defect detection, code quality, review speed, or developer productivity.

5. Conclusion and Future Work

This paper presented ReviewPilot, a proof-of-concept web platform for AI-assisted code review and pull request analysis. The proposed architecture integrates repository ingestion, pull request context, deterministic checks, configurable review rules, AI-generated candidate observations, issue grouping, review history, and dashboard-based indicators. The prototype interfaces demonstrate how these components can be organised within a unified human-in-the-loop workflow.

The principal contribution is the integration and articulation of the workflow rather than a validated claim of superior review performance. The current manuscript does not provide a labelled evaluation dataset, expert ground truth, baseline comparison, precision, recall, false-positive rate, response-time distribution, security assessment, or developer usability study. In addition, LLM-generated review comments may be irrelevant or hallucinated, and industrial adoption can create both benefits and workflow costs. ReviewPilot should therefore be treated as decision support, with deterministic evidence and human verification retained for every recommendation.

Future work should implement language-specific analysers, secure repository connectors, role-based access control, model and prompt versioning, comment provenance, CI/CD integration, and configurable data-retention policies. Evaluation should use public and project-specific pull request datasets with expert-labelled defects and review comments. Comparative experiments should measure precision, recall, F1-score, false-positive rate, severity agreement, latency, computational cost, and reviewer acceptance. Controlled developer studies should additionally examine review time, cognitive workload, comment usefulness, trust calibration, and the proportion of recommendations accepted or rejected. These steps are required to determine whether the proposed system provides reproducible benefits beyond existing static-analysis and pull request tools.

References

[1] A. Bacchelli, C. Bird, (2013). Expectations, outcomes, and challenges of modern code review. In 2013 35th international conference on software engineering (ICSE), IEEE, San Francisco, CA, USA. https://doi.org/10.1109/ICSE.2013.6606617

[2] S. McIntosh, Y. Kamei, B. Adams, A.E. Hassan, (2014). The impact of code review coverage and code review participation on software quality: A case study of the Qt, VTK, and ITK projects, in Proc. 11th Working Conf. Mining Software Repositories (MSR), 192–201. https://doi.org/10.1145/2597073.2597076

[3] C. Sadowski, E. Söderberg, L. Church, M. Sipko, A. Bacchelli, (2018). Modern code review: A case study at Google, in Proc. 40th Int. Conf. Software Engineering: Software Engineering in Practice (ICSE-SEIP), 181–190. https://doi.org/10.1145/3183519.3183525

[4] Z. Yang, C. Gao, Z. Guo, Z. Li, K. Liu, X. Xia, and Y. Zhou, A survey on modern code review: Progresses, challenges and opportunities. arXiv preprint arXiv:2405.18216. https://doi.org/10.48550/arXiv.2405.18216

[5] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, N. Sundaresan, (2022). Automating code review activities by large-scale pre-training, in Proc. 30th ACM Joint European Software Engineering conference and symposium. Foundations of Software Engineering (ESEC/FSE), 1035-1047. https://doi.org/10.1145/3540250.3549081

[6] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, (2022) Using pre-trained models to boost code review automation, In Proceedings of the 44th international conference on software engineering, 2291-2302. https://doi.org/10.1145/3510003.3510621

[7] J. Lu, L. Yu, X. Li, L. Yang, C. Zuo, (2023). Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, Florence, Italy. https://doi.org/10.1109/ISSRE59848.2023.00026

[8] C. Pornprasit, C. Tantithamthavorn, (2024).Fine-tuning and prompt engineering for large language models-based code review automation. Information and Software Technology, 175, 107523. https://doi.org/10.1016/j.infsof.2024.107523

[9] A. Naik, M. Alenius, D. Fried, C. Rose, (2025). CRScore: Grounding automated evaluation of code review comments in claims and code changes. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, Albuquerque, New Mexico. https://doi.org/10.18653/v1/2025.naacl-long.457

[10] C. Liu, H.Y. Lin, P. Thongtanunam, (2025), December. Hallucinations in Code Change to Natural Language Generation: Prevalence and Evaluation of Detection Metrics. In Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, Mumbai, India. https://doi.org/10.18653/v1/2025.ijcnlp-long.137

[11] U. Cihan, V. Haratian, A. İçöz, M.K. Gül, Ö. Devran, E.F. Bayendur, B.M. Uçar, E. Tüzün, (2024). Automated code review in practice. In Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), IEEE, Ottawa, ON, Canada. https://doi.org/10.1109/ICSE-SEIP66354.2025.00043

[12] H.Y. Lin, P. Thongtanunam, C. Treude, W. Charoenwet, (2024). Improving automated code reviews: Learning from experience. In Proceedings of the 21st International Conference on Mining Software Repositories, 278-283. https://doi.org/10.1145/3643991.3644910

Funding

No funding was received for conducting this study.

Conflict of interest

The Author's have no conflicts of interest to declare that they are relevant to the content of this article.

About The License