International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Autoshield AI: An AI-Powered Code Review System

Chetan Rakhade, Mayur Khodankar, Purnanshu Thakare, Achal Kale, Bramesh Gajbhiye, Prajwal Gaidhane

Department of Computer Science and Engineering, School of Engineering and Technology
G H Raisoni University, Amravati, Maharashtra, India · Academic Year 2026–2027

Abstract— Code review supports software quality by helping identify potential programming errors, security concerns, performance issues, and code-quality problems. Manual review remains important but can require time and technical expertise. This paper presents AutoShield AI, a web-based application developed to assist developers and students with initial source-code analysis. Authenticated users can submit source-code files and receive AI-generated feedback about possible bugs, security vulnerabilities, performance, code quality, and improvements. The system also combines review-history management, source-code-based interview-question generation, and PDF report generation in one application. The frontend uses React.js and Vite; backend services use Node.js and Express.js; MongoDB and Mongoose provide data storage; JSON Web Tokens support authentication; and the Groq API provides the external AI-analysis service. The thesis reports 31 functional test cases across general functionality, authentication, code review, interview questions, and PDF reporting, all marked as passed. These results represent functional checks rather than an independent measurement of AI accuracy or comparative performance. AutoShield AI demonstrates an integrated implementation of AI-assisted code-review and learning-support features, while its recommendations still require human verification.

Keywords— Artificial intelligence, code review, source-code analysis, software quality, developer learning, Groq API.

I. INTRODUCTION

Code review is used in software development to identify defects, discuss implementation choices, and improve maintainability. Manual inspection can provide context-sensitive feedback, but it depends on reviewer availability and experience. Research on modern code review has examined its relationship with software quality and review practices [8], while recent work has explored automating parts of review activity [10], [14].

Advances in pretrained models for code and natural language have enabled systems to generate code-related representations, explanations, and suggestions [1], [5], [16], [17]. These capabilities can support initial review, but generated outputs are not guaranteed to be correct and should not be treated as a substitute for testing or expert review.

AutoShield AI is a web application that combines AI-assisted source-code review with user authentication, review history, interview-question generation, and PDF reporting. The contribution described in this paper is the integration and implementation of these functions in one application; this work does not propose a new foundation model or claim a new code-analysis algorithm.

The remainder of the paper reviews relevant background, describes the architecture and workflow, summarizes implementation and functional testing, and discusses limitations and future work.

II. RELATED WORK

A. Models for Code Understanding

Research has explored language models trained on code, including evaluations of code-generation capability [1], coding benchmarks such as APPS [2], and pretrained models such as CodeBERT [5]. Further work has examined graph-based representations of programs [7], code summarization [3], code generation [4], meaningful code changes [6], and unified models for code understanding and generation [16], [17]. Studies of code duplication in machine-learning models [15] also highlight that code representation and data characteristics matter. Related model literature includes the GPT-4 technical report [11], few-shot language models [12], StarCoder [13], Code Llama [18], the Transformer architecture [19], and BERT [20]. These studies provide context for the broader use of machine learning in software engineering. AutoShield AI does not train or fine-tune these models; it sends code to an external AI service through an API.

B. Code Review and Automated Assistance

Empirical research has studied the effects of modern code-review practices on software quality [8]. Work on automating code-review activities and using pretrained models to support review tasks is directly relevant to AI-assisted development workflows [10], [14]. Static-analysis tools, including platforms such as SonarQube [9], support code-quality checks, although rule-based findings and AI-generated explanations serve different roles. AutoShield AI focuses on presenting AI-generated feedback alongside supporting application features rather than replacing static-analysis tools.

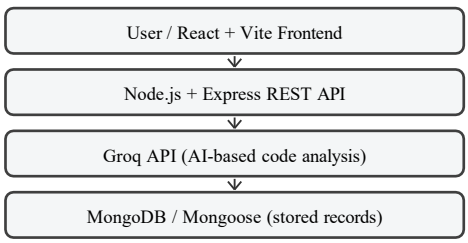
C. Identified Application Need

The project thesis identifies a practical integration need: users may need separate tools for code analysis, saved reviews, interview preparation, account management, and report creation. AutoShield AI addresses this application-level need by bringing these capabilities together. The current evaluation documents whether the implemented functions operate; it does not compare the system against other tools or measure defect-detection precision and recall.

III. SYSTEM DESIGN AND METHODOLOGY

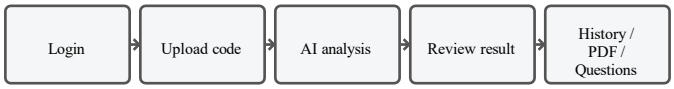
A. Architecture

The application follows a client-server architecture. The React.js and Vite frontend provides screens for registration, login, code submission, review results, history, profile management, interview questions, and reports. The Node.js and Express.js backend receives REST requests, applies authentication checks, coordinates database operations, and calls the AI service. MongoDB stores user and review-related records, while Mongoose supports data modelling and database access. JWT-based authentication protects restricted routes and associates records with the authenticated user.



B. Code-Review Workflow

After authentication, a user uploads a source-code file through the frontend. The backend validates the request and forwards the relevant code to the Groq API for analysis. The service returns generated feedback, which the backend passes to the frontend for display. The application stores review information so the user can later retrieve or delete a review. Separate features generate interview questions from submitted code and create a PDF version of review results.



Conceptual workflow derived from the implemented application

Fig. 2. High-level user workflow.

C. Review Output and Supporting Features

The review is intended to provide structured feedback about potential bugs, security concerns, performance, code quality, and possible improvements. Since the AI service may produce incomplete or incorrect content, users should inspect suggestions before changing source code. Review history supports retrieval of previous results; interview-question generation is intended for learning and preparation; PDF output supports saving and sharing a review summary.

D. Technology Stack

The thesis records React.js and Vite for the frontend; JavaScript, Node.js, and Express.js for backend application logic; MongoDB and Mongoose for persistence; JWT for authentication; REST APIs for frontend-backend communication; and the Groq API using an OpenAI-compatible model interface for AI analysis. Git/GitHub and Hoppscotch are listed as development and API-testing tools. The thesis names the configured model as openai/gpt-oss-120b; this is an integration choice rather than a model developed by the project team.

IV. IMPLEMENTATION

The implementation organizes user-facing functionality around the authenticated application workflow. Account creation and login provide access to protected features. Code-review requests pass from the frontend to the backend, which handles the request and communicates with the external AI service. The returned review is displayed in the user interface and persisted for later access. The history module supports viewing and deleting review records. The interview-question module generates questions related to submitted source code, while the PDF module creates a portable report from review information.

This architecture separates presentation, request handling, persistence, authentication, and AI-service communication. The separation makes each component easier to maintain, although reliability of AI review remains dependent on the external service and the quality of its generated response.

V. TESTING AND RESULTS

The project thesis reports functional testing for the main application modules. The reported test cases cover registration and login, code upload and review, review history and deletion, interview-question generation, profile display, logout, and PDF report creation. Table I summarizes the counts reported in the thesis. All statuses in the thesis summary are marked “Pass.”

Module	Test cases	Reported result
Functional testing	10	Pass
Authentication	5	Pass
Code review	6	Pass
Interview questions	5	Pass
PDF report	5	Pass
Total	**31**	**All marked Pass**

Table I. Functional test summary reproduced from the project thesis.

These results indicate that the thesis documents successful completion of the listed functional checks in its test environment. They do not establish the accuracy of AI-generated bug reports, the rate of missed vulnerabilities, response latency, or superiority over other review tools. No quantitative benchmark dataset or comparative evaluation is reported in the source thesis, so such claims are not made here.

VI. LIMITATIONS AND FUTURE WORK

AutoShield AI depends on the availability of an external Groq API and an internet connection for AI-based analysis. Generated feedback may be incomplete or inaccurate and must be reviewed by a developer. The current application is an assistance tool and does not replace professional code review, static analysis, software testing, or security auditing. The thesis also identifies future work including broader programming-language support, improved handling of large code files, integration with static-analysis tools, prompt/model improvements, repository integration, continuous code-quality monitoring, and expanded learning/reporting features.

VII. CONCLUSION

This paper described AutoShield AI, a web-based application integrating AI-assisted code review with authentication, review history, interview-question generation, and PDF reporting. The system uses a React/Vite frontend, Node.js/Express backend, MongoDB/Mongoose storage, JWT authentication, and an external Groq API for AI analysis. The project thesis reports 31 functional test cases across five categories, all marked as passed. These tests document the operation of application features but do not measure AI-review accuracy or comparative effectiveness. AutoShield AI therefore represents a practical integrated application for initial code-analysis assistance and developer learning, with human verification remaining essential.

ACKNOWLEDGMENT

The authors acknowledge the guidance provided by Ms. Trupti Kantale and the support of the Department of Computer Science and Engineering, School of Engineering and Technology, G H Raisoni University, Amravati.

REFERENCES

[1] M. Chen et al., “Evaluating Large Language Models Trained on Code,” arXiv:2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[2] D. Hendrycks et al., “Measuring Coding Challenge Competence With APPS,” Advances in Neural Information Processing Systems, vol. 34, 2021. [Online]. Available: https://arxiv.org/abs/2105.09938

[3] A. LeClair, S. Jiang, and C. McMillan, “Improved Code Summarization via a Graph Neural Network,” in Proc. 28th International Conference on Program Comprehension, 2020. doi: 10.1145/3387904.3389268.

[4] Y. Li et al., “Competition-Level Code Generation with AlphaCode,” Science, vol. 378, no. 6624, pp. 1092–1097, 2022. doi: 10.1126/science.abq1158.

[5] Z. Feng et al., “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” in Findings of ACL: EMNLP 2020, pp. 1536–1547, 2020. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.139/

[6] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk, “On Learning Meaningful Code Changes via Neural Machine Translation,” in Proc. 41st International Conference on Software Engineering, pp. 25–36, 2019. doi: 10.1109/ICSE.2019.00021.

[7] M. Allamanis, M. Brockschmidt, and M. Khademi, “Learning to Represent Programs with Graphs,” arXiv:1711.00740, 2017. [Online]. Available: https://arxiv.org/abs/1711.00740

[8] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, “An Empirical Study of the Impact of Modern Code Review Practices on Software Quality,” Empirical Software Engineering, vol. 21, pp. 2146–2189, 2016. doi: 10.1007/s10664-015-9381-9.

[9] G. A. Campbell and P. P. Papapetrou, SonarQube in Action. Manning Publications, 2013.

[10] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, “Towards Automating Code Review Activities,” in Proc. 43rd International Conference on Software Engineering, 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/towards-automating-code-review-activities/

[11] OpenAI, “GPT-4 Technical Report,” arXiv:2303.08774, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[12] T. B. Brown et al., “Language Models Are Few-Shot Learners,” Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[13] R. Li et al., "StarCoder: May the Source Be With You!" Transactions on Machine Learning Research, 2023. [Online]. Available: https://arxiv.org/abs/2305.06161

[14] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using Pre-Trained Models to Boost Code Review Automation," arXiv:2201.06850, 2022. [Online]. Available: https://arxiv.org/abs/2201.06850

[15] M. Allamanis, "The Adverse Effects of Code Duplication in Machine Learning Models of Code," in Proc. ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, pp. 143–153, 2019. doi: 10.1145/3359591.3359735.

[16] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "CodeT5: Identifier-Aware Unified Pre-Trained Encoder-Decoder Models for Code Understanding and Generation," in Proc. EMNLP 2021, pp. 8696–8708, 2021. [Online]. Available: https://aclanthology.org/2021.emnlp-main.685/

[17] D. Guo et al., "GraphCodeBERT: Pre-Training Code Representations with Data Flow," in International Conference on Learning Representations, 2021. [Online]. Available: https://arxiv.org/abs/2009.08366

[18] B. Rozière et al., "Code Llama: Open Foundation Models for Code," arXiv:2308.12950, 2023. [Online]. Available: https://arxiv.org/abs/2308.12950

[19] A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[20] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, pp. 4171–4186, 2019. [Online]. Available: https://aclanthology.org/N19-1423/

Pre-submission note: *This manuscript is a draft assembled from the supplied thesis. Before submission, all bibliographic metadata and URLs should be checked against the original publications; author-name spellings and institutional details should be confirmed by all six authors; and the target journal's current template and ethics/plagiarism requirements should be checked. The manuscript intentionally does not claim measured AI accuracy or comparative performance because such results are not reported in the thesis.*