IJAIA Volume 1 | Number 2 | December 2025 | Pages 1-11

International Journal of Artificial Intelligence Applications

ISSN (online): 3105-3572
Homepage: ijaia.com

Implementing AI-Based Code Review Automation: A Case Study in Academic Software Development

Omar Isam AL Mrayat [1*], Dyala Ibrahim[2], Malik jawarneh[3]

[1]Department of Software Engineering, Amman Arab University,
11953 Amman, Jordan, Email : o.mrayat@aau.edu.jo
[2]Department of Cyber Security, Amman Arab University,
11953, Amman, Jordan, Email : d.ibrahim@aau.edu.jo
[3]Department of computer science, Amman Arab University,
11953 Amman, Jordan Email : m.jawarneh@aau.edu.jo

Received: 08, 2025 *Revised: 10, 2025* *Accepted: 11, 2025* *Available online: 12, 2025*

ABSTRACT — Code reviews have always been a necessary evil in software development but one that was very time consuming. Nevertheless, it is much tougher for educators in education due to evaluating codes for hundreds of students. This paper describes the design and deployment of an AI-assisted code review system in a university computer science department. The system uses the transformer and GPT-4 models for aiding the human graders in assessing programming assignments of various university courses. The system checks the code for syntax and semantic errors, design and implementation patterns, and compliance with coding standards. In a six-month study run of involving 150 student projects, the automated system detected around 87% of code quality issues needing human inspection. It cut down the time taken by educators to review code by approximately 62%, with the same grade of feedback quality. Based on our experience, AI for code review has exciting potentials but also currently has limitations in an academic environment. It is good at spotting common mistakes as well as style issues, but not so good at picking up on context-sensitive design decisions or pedagogical issues. The study provides evidence of AI-supported code review for the academic field and suggestions for educators wishing to set up a similar system.

Keywords — Artificial Intelligence; Code Review Automation; GPT-4; Software Engineering Education; Large Language Models; Automated Assessment; Educational Technology.

1. INTRODUCTION

All code should be reviewed by developers in software engineering practices in order to detect bugs, enforce coding standards, and improve quality. The manual procedure is tiring and takes a lot of time, at least in the universities where professors have heaps of stacks of assignments to read through. Do not forget the humans' inconsistency. Recent developments in AI, especially large-scale language models and transformer architectures create interesting new possibilities for automating at least parts of code review [1]. Recent AI-powered frameworks have demonstrated effectiveness in enhancing bug detection and code reliability through automated testing [2]. AI is increasingly being introduced in code review in universities and companies as well. Research shows that these tools can accurately identify code smells, security vulnerabilities, styling issues, among others. However, not much investigation has occurred of these artefacts specifically within educational settings. The teaching contexts arise several challenges – students of vastly different expertise, providing some feedback the learner can actually use, trying to keep human in the loop for pedagogical reasons, etc. This project highlights our experience with using AI-based code review system at university to

* Corresponding author

tackle these issues. We adapted recent large language models work, primarily focused on GPT-4, to fit the requirements for academic code grading [3].

The integration of AI techniques is done in our system. Thus, it generates a comment using natural language processing. Moreover, it recognizes patterns using machine learning. Also, it uses the expert system which is rule-based for checking constraints. Our contributions involve: first, we build an AI-based code review system for academic use. Second, we evaluated the system over several programming courses with students with different programming skills. Third, we analyzed the impact on instructor effort and the quality of feedback. Fourth, we set the boundaries of the current deep learning AI model's capability for academic code reviews. Fifth, we provide actionable recommendations for educators and institutions planning such a system [4].

2. RELATED WORK

2.1. Automated Code Review Systems

Automated code review has come a long way over the last ten years or so. Traditional static analysis tools have gotten much smarter with machine learning capabilities for detecting complex patterns and potential bugs. Tufano and colleagues [1] showed that deep learning models can manage various code review tasks pretty well, including generating comments and assessing quality. Their analysis was comprehensive and revealed both what works and what does not in current approaches, making it clear we need to keep pushing forward in this area.

Vijayvargiya et al. [3] described Auto Commenter, an LLM-based system Google deployed for four programming languages. They proved you can do this at massive scale while keeping users happy. Similarly, AI-enhanced code review systems [5, 6, 7] and automated comment generation tools [8] demonstrated that systems built on GPT-4 can provide useful automated review comments across diverse types of code repositories. AI-powered testing frameworks [2] have shown particular promise in detecting defects through machine learning and anomaly detection techniques.

Yin et al. [4] suggested using program structure information through graph representations to make code review more effective. Their PDG2Seq algorithm turns program dependency graphs into sequences while keeping structural and semantic information intact, showing better results than approaches that just look at sequences.

2.2. AI and Large Language Models in Software Engineering

Turner [9] introduceed transformer architectures and their mathematical foundations. Islam et al. [10] conducted an extensive survey of transformer applications in deep learning tasks, highlighting their ability to model contextual relationships in sequential data. Recent research examines both the promises and perils of transformer-based models for software engineering [11].

The transformer architecture introduced by Vaswani et al. [12] forms the foundation of modern language models. GPT-4's technical capabilities [13] have been extensively evaluated on programming tasks, demonstrating strong performance on code generation challenges [14] and systematic code evaluation [15], though requiring human validation for production use [16].

2.3. Software Engineering Education and AI

More attention is recently being paid to artificial intelligence tools being integrated into software engineering education. Recent research has explored the potential applications of generative AI to enhance various aspects of computer science education (for example, personalized learning and automated assessment) [17, 18, 19]. Concerns about academic integrity and the pedagogical implications of AI help remain live issues.

According to Syeed et al. [20], undertaking outcome-based education is advantageous due to the realization of the industry-driven subject software engineering in practice. Project-based learning approaches [21] and integrated frameworks for educational software development [22] combined with digital learning in higher education [23] and DevOps integration into curricula [24, 25] support modern software engineering education. Petrovska et al. [17] conducted a competency-based assessment of generative AI in software development educational activities. Recent workshops [26] and studies on critical thinking skills [27] emphasize human-AI collaboration in curriculum development [28], with universities building capacity through open source programs [29].

3. METHODOLOGY

3.1. System Architecture

Our AI code review system has been designed to be modular and hence divided into four important components which are: The Code Analysis Engine, AI Inference Module, Feedback Generation System, Review Management Interface. The Code Analysis Engine conducts preliminary static analysis incorporating standard tools to extract metrics, identify syntax errors, flag common anti-patterns, and assess code quality attributes [30, 31]. Deep learning and object-oriented metrics [32] inform our analysis engine design. The initial iteration of this code, along with its generated results, is structured in a way that it can be supplied to AI models for verification and correction of complex semantic errors in the code. A basic level check will now occur automatically in the system for any student code. We optimized custom prompts for code review in an educational setting by using OpenAI API to tap into GPT-4 using AI Inference Module. We built different prompting templates for programming languages like python, java and others and similar templating assignment to do with algorithms, data structure, web dev. The API in the module makes batch calls, optimizes token pattern, while attempts to maintain context in file's chunk across file chunk. The system for generating feedback combines the findings from static analysis and AI inferences to create together a code review comment. The comments filtering module removes repetitive comments, highlights problems, and formats comments to improve pedagogical use. We incorporated educational best practices like constructive tone, specific examples, actional recommendations, and more. The Review Management Interface provides instructor tools to define review criteria, counter-check review performance, validate feedback from AI, and export results for further grading. The interface enables teachers to reject the machine's suggestions, and the changes to the question generation system are documented in order to train it further. The system architecture and inter-component data flow are shown in Figure 1.

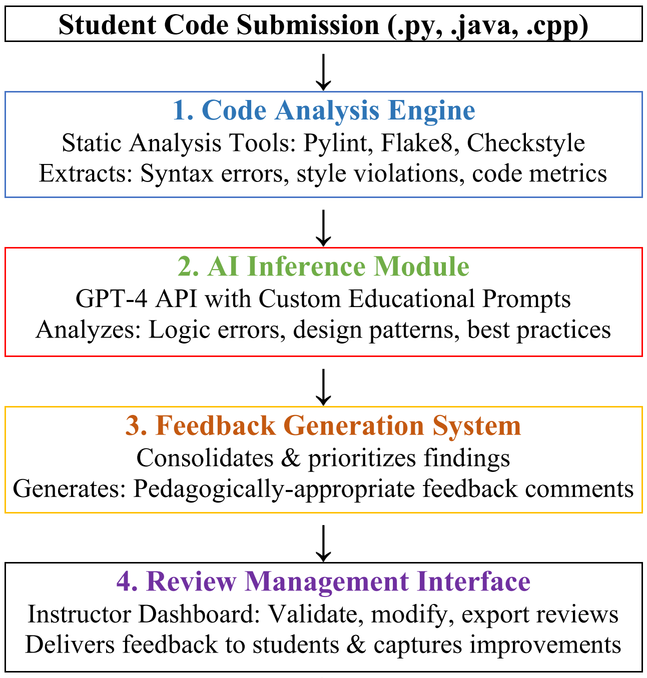
Student Code Submission (.py, .java, .cpp)
1. Code Analysis Engine
Static Analysis Tools: Pylint, Flake8, Checkstyle
Extracts: Syntax errors, style violations, code metrics
2. AI Inference Module
GPT-4 API with Custom Educational Prompts
Analyzes: Logic errors, design patterns, best practices
3. Feedback Generation System
Consolidates & prioritizes findings
Generates: Pedagogically-appropriate feedback comments
4. Review Management Interface
Instructor Dashboard: Validate, modify, export reviews
Delivers feedback to students & captures improvements

Figure 1. System Architecture Overview

3.2. Implementation Details

The official Python library from OpenAI allows easy access to the GPT-4 model. Our implementation considers efficient deep learning inference techniques [33]. The following are the main dependencies we used (except standard python libraries) that are required to use GPT-4. A complete system was developed utilizing python. The official Python library from OpenAI allows easy access to the GPT-4 model. To break down your code for discovering static errors we used Pylint and Flake8. The library Pandas Python Data have been used to parse data. We have deployed all components on university servers after implementing all the security features for keeping any student code safe from any misuse and also to prevent plagiarism. A few parameter settings in GPT-4 were tried out by us to set definitions for our system. We experimented with modifying the temperature setting and maximum tokens. We tried using various prompt texts and saw which one looked to be the most effective. We attempted different organization and grouping of our prompts in a variety of budgets. After several tries, we were able to discover effective setups for the different situation types. A temperature setting of 0.3 serves us well in consistently forcing GPT-4 to catch clear-cut objective errors. Figure 2 compares the traditional manual review workflow with our AI-assisted approach, highlighting the time savings and consistency improvements achieved through automation of routine checks while maintaining human oversight for complex decisions.

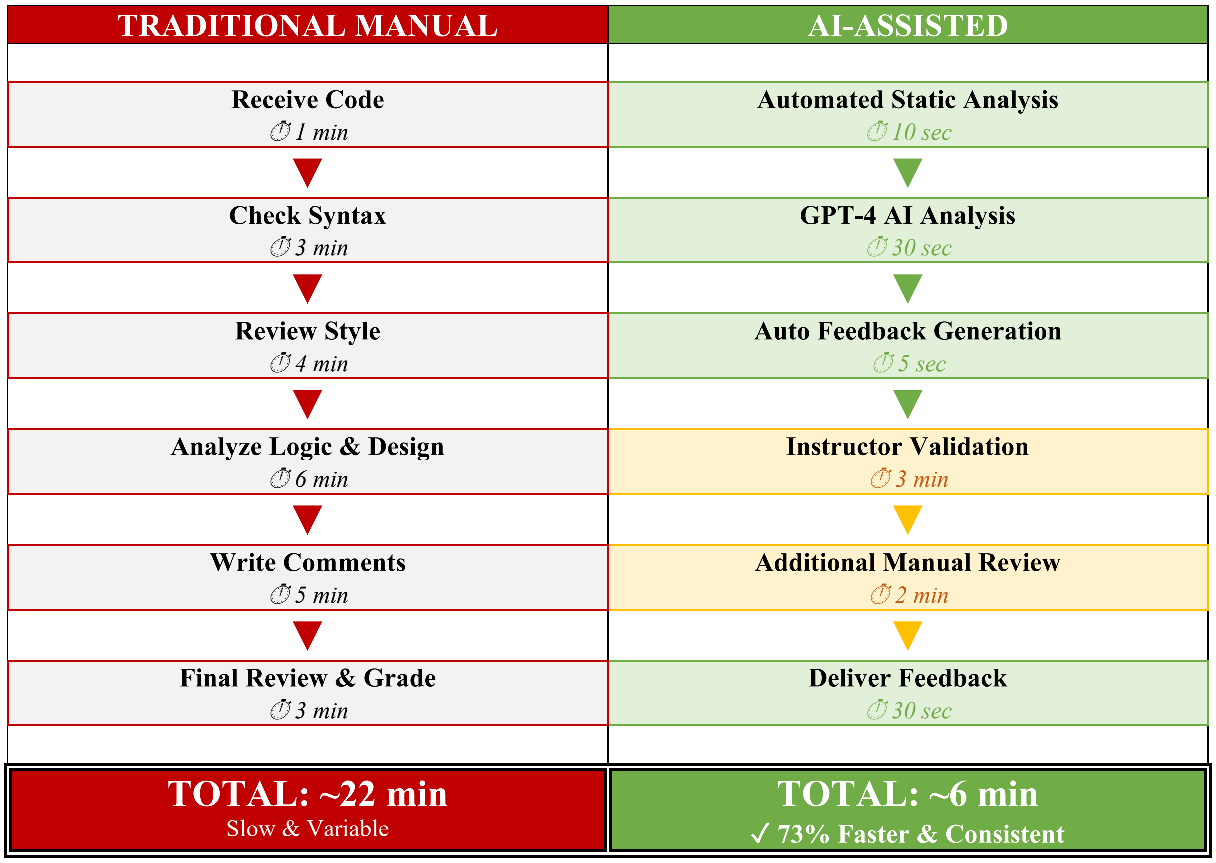

Figure 2. Traditional vs. AI-Assisted Code Review Workflow

3.3. Data Collection and Experimental Setup

The investigation was conducted in a period of 6 months, namely Spring and Fall semesters of 2024 in three computer science classes. The study sample comprised of 150 students. Across all the course assignments submitted, there were 450 assignments, many of which were simple algorithms to multi-file projects. To comprehensively evaluate our measures, we employ a combination of qualitative and quantitative techniques. We possess quantitative metrics on one side. Conversely, there are qualitative assessments. Measures such as detection accuracy (precision, recall, F1-score) for various issue types, review time reduction and consistency measures. The qualitative analysis consists of an instructor survey, student feedback analysis, and a case study of examples where recommendations differ from human judgments. Manual review is required to ensure our evaluation's ground truth.

3.4. Ethical Considerations

The researchers received institutional review board approval and obtained informed consent from all the students. We made it clear to students from the very beginning that AI-assisted review would not take over grading or replace human review. Students could ask for purely human review at any time. All code, grades and feedback data were anonymized for the purpose of the research. We established strict data protection processes. We clearly mentioned about being over trusting the system for example, humans were always the ones to create final letter grades and make re-marking decisions. Furthermore, the students could question/challenge feedback from human instructors. Students were informed that this AI system is a software assistant that allows instructors to offer more.

4. RESULTS

4.1. Detection Accuracy

The AI system is significantly capable of determining the usual kinds of code quality problems. The detection accuracy for the distinct categories is shown in Table I. Detection rates for syntax errors and style violations are the best (precision: 0.94, recall: 0.91) as they are driven by high-coverage rules which are usually well-defined. Taking into consideration the high precision of 0.78 and recall of 0.71, logic errors and algorithmic inefficiencies are more difficult.

Table 1: DETECTION ACCURACY BY ISSUE CATEGORY

Issue Category	Precision	Recall	F1-Score
Syntax Errors	0.94	0.91	0.92
Style Violations	0.93	0.89	0.91
Missing Error Handling	0.86	0.84	0.85
Variable Naming Issues	0.91	0.87	0.89
Inefficient Loops	0.84	0.80	0.82
Logic Errors	0.78	0.71	0.74
Algorithmic Inefficiency	0.75	0.68	0.71
OOP Design Violations	0.69	0.65	0.67

The system demonstrated good classification performance, with an F1-score of 0.85 for missing error handling, 0.89 for inappropriate variable names, and 0.82 for inefficient loop constructs. In the case of object-oriented design principal violations, it showed a moderate performance (F1-score: 0.67), as that would require a holistic understanding of the entire system design. A comparison analysis revealed that the AI discovered about 87% of the things identified by the expert reviewers. However, it also caught an additional 15% legitimate problems that were not picked up by the human review.

4.2. Review Time Reduction

The instructor spent the following time on code review before and after AI Assisted code review. Refer the Table II. The time allocated to code review experiences a sharp decline. The faculty spent an average of 12 mins per assignment for the intro category and 25 mins per assignment for the advanced category on code review in total time before the implementation. On the contrary, that might use AI aided.

Table 2: REVIEW TIME COMPARISON (MINUTES PER ASSIGNMENT)

Course Level	Manual Review	AI-Assisted	Time Saved	Reduction
Introductory (CS101)	12.0	4.5	7.5	62%
Intermediate (CS201)	18.0	7.2	10.8	60%
Advanced (CS301)	25.0	9.5	15.5	62%
Simple Assignments	8.0	2.0	6.0	75%
Complex Projects	32.0	18.0	14.0	44%

According to the research, the time savings depend on the complexity of the contribution and also functional requirements and the skills of the students. For example, instructor review times were reduced 75% on average for assignments where the main problems were related to syntax and style. That was the easiest of contributions. Reduction of review times was lesser (40-45%) for projects requiring architecture scrutiny. The instructor still needs enough time to address the higher-level considerations for assessment. On the presenter end, although less time was consumed, the nature of the feedback was the same or better. The AI system flagged the issue in all the studies.

4.3. Feedback Quality and Consistency

The 150 student surveys revealed that the AI-assisted feedback received an overall positive assessment. The ratings are summarized in Table III. The feedback was rated clear by the students with a rating of 4.1. It was rated helpful at 3.9. Also, the feedback specificity was rated 4.2. According to qualitative comments, the students liked the feedback because it was.

Table 3: STUDENT FEEDBACK RATINGS (5-POINT LIKERT SCALE)

Aspect	Mean Score	Std Dev	Median
Clarity	4.1	0.7	4.0
Helpfulness	3.9	0.8	4.0
Specificity	4.2	0.6	4.0
Consistency	4.3	0.6	4.5
Timeliness	4.5	0.5	5.0
Overall Satisfaction	4.0	0.7	4.0

As analyzed for consistency, the feedback standard provided has been similar due to feedback by AI. Meaning that similar errors had similar feedback, regardless of when submitted and which TA assigned. This uniformity aids in standardizing consistency for large courses that could have assorted styles will there be different TAs. However, some students were opposed to the automation that took the human touch away. They wanted insight from their mistakes and not just the usual automated message. Students prefer context-specific feedback they think would benefit them more. Also, Promoted.

4.4. System Limitations and Failure Cases

The research uncovered system limitations that trendily cut across assignments: First, domain or context-specific requirements that are not mentioned in code comments or assignment hand-outs often tend to be out-of-scope as far as students are concerned and do not get checked by the system. For instance, one of the assignments requires that the system design follows the MVC pattern, and all the solutions followed that. The design was not vetted as it was only referred to the class lectures. Even if a different student produced a solution that took a different architecture, that also would have been a correct solution. However, it is imperative.

5. DISCUSSION

5.1. Practical Implications for Academic Institutions

As a result, institutions adopting generative AI for software practitioners [34] should expect to incur implementation costs upfront and have ongoing costs for API usage, prompt engineering, and system refinement. The transformation through AI agents [35] and the future of AI-driven software engineering [36, 37, 38] suggest early adoption provides competitive advantages. The deployment of AI-assisted code review in our setting was successful, indicating that it is practical with low instructional resources for a new institute. The system worked best when it was embedded in existing systems and workflows and did not replace a human reviewer completely. As a result, a school utilizing ChatGPT for real applications should expect to incur implementation costs upfront and expect to have ongoing costs of using the API plus time spent re-engineering prompts and reworking. Due to our experience, we know repetitive feedback on simple issues can lead to enhanced learning. This means AI-assisted review works best in a large intro course. For higher learning where the design and architecture matter more, the AI can be a first-pass filter for which the instructor can be the 'reviewer.' The cost of the API for GPT-4 is about $0.15 per assignment, and so far, the cost-benefit analysis indicated that we managed to recoup the costs through a reduction in grading assistant hours from first semester. Schools must take into consideration both direct costs.

5.2. Pedagogical Considerations

It is equally possible that the over-reliance on automated marking will slightly impair the effectiveness of certain types of learner activities, such as critical analysis and evaluation, if students take the feedback [39]. Using AI to review code raises big questions and challenges about what we want out of learning and its assessment authenticity. Our observations demonstrate that automated feedback can help students acquire basic programming skills, while it frees up the teacher for issues of concept and solution strategy. As our students got familiar with the basics of programming, instant feedback on syntactic and stylistic errors helped them understand the errors better and rectify them before showing the final submission. To put it differently, students learn from mistakes and get a chance to resubmit their assignment after correction. The current pedagogy of formative assessment and learning from one's mistakes was coordinated with this. It is equally possible that the over-reliance on automated marking will slightly impair the effectiveness of certain types of learner activities, such as critical analysis and evaluation, if students take the feedback.

5.3. Technical Insights on AI Model Performance

We also saw that prompt engineering played a key role in getting the best out of GPT-4 [40]. The model benefits from explicit role definitions and structured prompts. While transformer models show promise across domains [41], their application to software engineering requires domain-specific considerations [11]. The capabilities of the Model GPT-4 and its performance of the model were revealed in a study. To begin with, the model showed a deep understanding of the syntax and semantics of several programming languages including Python, Java and C++. As the model has been trained on a large set of public code repositories, it can identify common patterns and anti-patterns in code. We also saw that prompt engineering played a key role in getting the best out of GPT-4. The model will benefit from being given an explicit definition of roles, according to our code.

6. RECOMMENDATIONS AND FUTURE WORK

Based on our experiences, we have several suggestions for teachers and educational institutions that want to experiment with AI-assisted code-review. Start with the lessons and have simple tasks before starting on the big projects. Dedicate some time to prompt engineering: You do need to spend some time producing prompts for different courses, assignment types, and learning goals. Always ensure humans participate in the loop. AI can be used the direction of future research include development of educational code specialization models. Also training of models on code from student submissions and related learning feedback (explanations, hints). This could utilize advances in research on end user program repair. Reverse Engineering IDEs' Runtime Behavior through Their JVM Bytecode/Executable. The goal is to reveal the operation of an IDE's mechanisms at runtime through taking its bytecode or executable apart through static or dynamic means. Researching the extended effects that methods have on students' skills - Evidence based studies on the students' learning outcomes and programming skills.

7. Conclusion

The case study discussed here highlights feasibilities and practicalities of automation of student code review using AI based system. The system was implemented as a pilot in academic software development with 150 students from three different courses for six months. According to the study, employing AI to review code cuts down time usage for during code-reviewing by 62%. In addition, the responses from the instructors, students and the review of responses of the system were positive.

The system was able to find 87% of the problems in the student code, comparable to recent AI-powered frameworks for bug detection [2]. Also, its managed syntax errors, style errors, and common logical errors well. However, it could not recognize problems associated with context and design decisions. This means that the power of the human reviewer can be combined with automation which has the ability to scale. Consistent feedback can thus be provided on wrongly defined errors by AI.

Consequently, the teacher can concentrate more on advanced-dimension education interactions and difficult evaluation. Likewise, the teacher must integrate human assessment into the system to give classes on design and context issues. Issues related to sensitive situations and equity are also managed by humans. AI provides a complementary function to the reviewer's human expertise. As AI continues to evolve, future systems will address more complex challenges through multi-modal data and analysis of participant responses. Just like that, AI contribution. Students and educators must also utilize smarter software, as software engineers are doing so. Through this paper we hope to ignite more research in the area.

REFERENCES

[1] R. Tufano, O. Dabić, A. Mastropaolo, M. Ciniselli, and G. Bavota, "Code Review Automation: Strengths and Weaknesses of the State of the Art," IEEE Trans. Softw. Eng., vol. 50, no. 2, pp. 338–353, Feb. 2024. DOI: 10.1109/TSE.2023.3348172

[2] O. I. Al Mrayat, M. Jawarneh, D. Ibrahim, and A. Altrad, "AI-Powered Software Testing: A Novel Framework for Enhancing Bug Detection and Code Reliability," Int. J. Intell. Syst. Appl. Eng., vol. 12, no. 23s, pp. 1871–, Sep. 2024. DOI: 10.17762/ijisae.v12i23S.7147

[3] M. Vijayvergiya et al., "AI-Assisted Assessment of Coding Practices in Modern Code Review," in Proc. AIware '24, Porto de Galinhas, Brazil, Jul. 2024, pp. 1–9. DOI: 10.1145/3664646.3665664

[4] Y. Yin, Y. Zhao, Y. Sun, and C. Chen, "Automatic Code Review by Learning the Structure Information of Code Graph," Sensors, vol. 23, no. 5, p. 2551, Feb. 2023. DOI: 10.3390/s23052551

[5] K. Ye, L. Zhou, and S. Huang, "Automated Code Review In Practice," arXiv preprint arXiv:2412.18531, Dec. 2024.

[6] D. Groeneveld et al., "AICodeReview: Advancing Code Quality with AI-Enhanced Reviews," SoftwareX, vol. 26, p. 101677, 2024. DOI: 10.1016/j.softx.2024.101677

[7] R. Li et al., "Resolving Code Review Comments with Machine Learning," in Proc. 46th Int. Conf. Softw. Eng.: Softw. Eng. in Practice (ICSE-SEIP), Lisbon, Portugal, Apr. 2024. DOI: 10.1145/3639477.3639746

[8] Y. Li et al., "CodeDoctor: Multi-Category Code Review Comment Generation," Autom. Softw. Eng., vol. 32, no. 1, Feb. 2025. DOI: 10.1007/s10515-025-00491-y

[9] R. E. Turner, "An Introduction to Transformers," arXiv preprint arXiv:2304.10557, Feb. 2024.

[10] S. Islam et al., "A Comprehensive Survey on Applications of Transformers for Deep Learning Tasks," Expert Syst. Appl., vol. 241, p. 122666, May 2024. DOI: 10.1016/j.eswa.2023.122666

[11] T. Zuo, H. Zhang, and M. Kim, "Promises and Perils of Using Transformer-Based Models for SE Research," Neural Networks, vol. 182, p. 107067, Dec. 2024. DOI: 10.1016/j.neunet.2024.107067

[12] A. Vaswani et al., "Attention Is All You Need," in Proc. 31st Int. Conf. Neural Inf. Process. Syst., Red Hook, NY, USA, 2017, pp. 6000–6010.

[13] J. Achiam et al., "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, Mar. 2024.

[14] P. Vishnu, "Unveiling the Role of GPT-4 in Solving LeetCode Programming Problems," Comput. Appl. Eng. Educ., vol. 33, no. 1, Jan. 2025. DOI: 10.1002/cae.22815

[15] C. Song and H. Lin, "A Systematic Evaluation of Large Language Models for Generating Programming Code," Neural Networks, vol. 172, Mar. 2024. DOI: 10.1016/j.neunet.2024.106117

[16] R. Poldrack, "AI-Assisted Coding: Experiments with GPT-4," arXiv preprint arXiv:2304.13187, Apr. 2023.

[17] O. Petrovska, L. Clift, F. Moller, and R. Pearsall, "Incorporating Generative AI into Software Development Education," in Proc. 8th Conf. Comput. Educ. Practice, pp. 37–40, 2024. DOI: 10.1145/3647649.3647651

[18] B. A. Becker et al., "Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation," in Proc. 54th ACM Tech. Symp. Comput. Sci. Educ., vol. 1, pp. 500–506, Mar. 2023. DOI: 10.1145/3545945.3569759

[19] A. Nguyen et al., "Software Engineering Education in the Era of Conversational AI: Current Trends and Future Directions," Front. Artif. Intell., vol. 7, Jul. 2024. DOI: 10.3389/frai.2024.1436350

[20] M. M. Syeed, A. Shihavuddin, M. F. Uddin, M. Hasan, and R. H. Khan, "Outcome Based Education (OBE): Defining the Process and Practice for Engineering Education," IEEE Access, vol. 10, pp. 119170–119192, 2022. DOI: 10.1109/ACCESS.2022.3219477

[21] E. Ceh-Varela, C. Canto-Bonilla, and D. Duni, "Application of Project-Based Learning to a Software Engineering Course in a Hybrid Class Environment," Inf. Softw. Technol., vol. 158, p. 107189, 2023. DOI: 10.1016/j.infsof.2023.107189

[22] A. Mbiada, B. Isong, F. Lugayizi, and A. Abu-Mahfouz, "Towards Integrated Framework for Efficient Educational Software Development," in 2023 IEEE/ACIS 21st Int. Conf. Softw. Eng. Res., Manag. Appl. (SERA), pp. 53–60, May 2023. DOI: 10.1109/SERA57763.2023.10197734

[23] M. Alenezi, "Digital Learning and Digital Institution in Higher Education," Educ. Sci., vol. 13, no. 1, p. 88, 2023. DOI: 10.3390/educsci13010088

[24] J. Díaz et al., "Harmonizing DevOps Taxonomies – A Grounded Theory Study," J. Syst. Softw., vol. 208, p. 111908, 2024. DOI: 10.1016/j.jss.2023.111908

[25] E. Sarmiento-Calisaya, A. Mamani-Aliaga, and J. C. Leite, "Introducing Computer Science Undergraduate Students to DevOps Technologies from Software Engineering Fundamentals," in Proc. ICSE 2024 Softw. Eng. Educ. Train., Apr. 2024.

[26] S. S. Rathore, S. Tiwari, and S. U. Farooq, "Workshop Report on Emerging Software Engineering Education," in Proc. 17th Innov. Softw. Eng. Conf. (ISEC '24), Feb. 2024, pp. 1–2. DOI: 10.1145/3641399.3641419

[27] J. Wynekoop and K. Nakatani, "Critical Thinking Skills for Computing Professionals: Closing the Education–Industry Gap," Ind. High. Educ., vol. 38, no. 4, pp. 376–384, 2024. DOI: 10.1177/09504222231219464

[28] A. Padovano and M. Cardamone, "Towards Human-AI Collaboration in the Competency-Based Curriculum Development Process: The Case of Industrial Engineering and Management Education," Comput. Educ.: Artif. Intell., vol. 7, p. 100256, 2024. DOI: 10.1016/j.caeai.2024.100256

[29] J. Morrison et al., "Building Software Engineering Capacity through a University Open Source Program Office," in FSE Companion '24, Porto de Galinhas, Brazil, Jul. 2024. DOI: 10.1145/3663529.3663866

[30] U. Iftikhar, N. Ali, J. Börstler, and M. Usman, "A Tertiary Study on Links Between Source Code Metrics and External Quality Attributes," Inf. Softw. Technol., vol. 165, p. 107348, Jan. 2024. DOI: 10.1016/j.infsof.2023.107348

[31] N. Nikolaidis, N. Mittas, A. Ampatzoglou, D. Feitosa, and A. Chatzigeorgiou, "A Metrics-Based Approach for Selecting Among Various Refactoring Candidates," Empir. Softw. Eng., vol. 29, no. 1, Dec. 2023. DOI: 10.1007/s10664-023-10412-w

[32] A. Tete, F. Toure, and M. Badri, "Using Deep Learning and Object-Oriented Metrics to Identify Critical Components in Object-Oriented Systems," in Proc. 2023 5th World Symp. Softw. Eng., pp. 48–54, Sep. 2023. DOI: 10.1145/3631991.3631998

[33] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review," Proc. IEEE, vol. 111, no. 1, pp. 42–91, Jan. 2023. DOI: 10.1109/JPROC.2022.3226481

[34] C. Ebert and P. Louridas, "Generative AI for Software Practitioners," IEEE Softw., vol. 40, no. 4, pp. 30–38, Jul. 2023. DOI: 10.1109/MS.2023.3265877

[35] A. Panyam and P. Gujar, "How AI Agents Are Transforming Software Engineering and the Future of Product Development," Computer, vol. 58, no. 5, pp. 71–77, May 2025. DOI: 10.1109/MC.2024.3488378

[36] R. M. Cantalapiedra, V. Gutiérrez, and M. Á. Serrano, "The Future of AI-Driven Software Engineering," ACM Trans. Softw. Eng. Methodol., Dec. 2024. DOI: 10.1145/3715003

[37] P. Ardimento and M. Scalera, "Artificial Intelligence for Software Engineering: The Journey So Far and the Road Ahead," ACM Trans. Softw. Eng. Methodol., Dec. 2024. DOI: 10.1145/3719006

[38] L. Rossi and A. Marino, "From Today's Code to Tomorrow's Symphony: The AI Transformation of Developer's Routine by 2030," ACM Trans. Softw. Eng. Methodol., vol. 34, no. 2, Jan. 2025. DOI: 10.1145/3709353

[39] M. Alami, M. Zahedi, and O. Krancher, "The Role of Psychological Safety in Promoting Software Quality in Agile Teams," Empir. Softw. Eng., vol. 29, no. 5, Jul. 2024. DOI: 10.1007/s10664-024-10512-1

[40] M. Escobar, R. Noel, and O. Pastor, "Prompt-Guided Evaluation of GPT-4o, Gemini and DeepSeek on UML-to-Java Code Generation," in Advances in Conceptual Modeling. ER 2025. Lecture Notes in Computer Science, vol. 16190. Springer, Cham. DOI: 10.1007/978-3-032-08620-4_13

[41] S. Karki, J. Rjoub, and A. Bentahar, "Transformer Models in Biomedicine," BMC Med. Inform. Decis. Mak., vol. 24, no. 1, p. 221, Aug. 2024. DOI: 10.1186/s12911-024-02600-5