AUTOMATED CHECKING OF PROGRAMMING TASKS

Nafasov Mirzomurod Mukhamadovich
Akhtamova Laziza Azam qizi
Bukhara engineering technological institute, (Uzbekistan)

Abstract

The article presents a comparative analysis of the automation of programming assignment checking, its efficiency, accuracy and comparison with manual checking. It examines various assessment methods, their impact on student learning and implementation issues. By analyzing existing automated assessment systems, the study discusses the optimization of the process of checking practical programming assignments based on technology-based assessment strategies.

Keywords: automated grading, programming education, code evaluation, artificial intelligence, machine learning, static analysis, dynamic analysis, plagiarism detection, assessment efficiency.

Аннотация

В статье представлен сравнительный анализ автоматизации проверки заданий по программированию, ее эффективности, точности и сравнения с ручной проверкой. В ней рассматриваются различные методики оценки, их влияние на обучение студентов и проблемы внедрения. Анализируя существующие автоматизированные системы оценки, в исследовании обсуждается оптимизация процесса проверки практических заданий по программированию на основе стратегий оценки, основанных на технологиях.

Ключевые слова: автоматизированная проверка, обучение программированию, проверка кода, искусственный интеллект, машинное обучение, статический анализ, динамический анализ, обнаружение плагиата, эффективность оценки.

Annotatsiya

Maqolada dasturlashga oid topshiriqlarini tekshirishni avtomatlashtirishni, uning samaradorligi, aniqligi va qo'lda tekshirishga nisbatan qiyosiy tahlili keltirilgan. Unda turli baholash metodologiyalari, ularning o‘quvchilar bilimiga ta’siri va amalga oshirishdagi qiyinchiliklar o‘rganiladi. Mavjud avtomatlashtirilgan baholash tizimlarini tahlil qilish orqali tadqiqot texnologiyaga asoslangan baholash strategiyalari asosida dasturlash bo'yicha amaliy topshiriqlarni tekshirish jarayonini optimallashtirish haqida so'z boradi.

Kalit so'zlar: avtomatlashtirilgan tekshirish, dasturlash, kodni tekshirish, sun'iy intellekt, mashinali o'rganish, statik tahlil, dinamik tahlil, plagiatni aniqlash, baholash samaradorligi.

Introduction. Programming assignments serve as a fundamental component of computer science education, fostering students' problem-solving abilities, logical reasoning, and coding proficiency. These assignments not only reinforce theoretical concepts but also cultivate hands-on experience in algorithm design and software development. However, the manual evaluation of programming tasks presents several challenges, including time consumption, subjective grading, and the possibility of human errors. As the number of students in computer science courses continues to grow, the demand for efficient and scalable assessment methods becomes increasingly critical.

Automated assessment systems provide a promising solution by enabling instant feedback, ensuring consistency in grading, and reducing instructors' workload. These systems leverage various techniques, including static code analysis, dynamic testing, and machine learning-based evaluation, to assess students' submissions based on correctness, efficiency, coding style, and adherence to best practices. Additionally, automated feedback mechanisms can enhance the learning experience by allowing students to iteratively improve their code based on real-time suggestions.

Methods. A systematic review of existing automated grading tools and frameworks was conducted to analyze the state-of-the-art approaches in programming assignment evaluation. This review encompassed a comprehensive examination of various assessment methodologies, including static analysis, dynamic analysis, and machine learning-based evaluation. Each method was assessed based on its ability to measure code correctness, computational efficiency, and adherence to coding standards.

To further investigate the practical implementation and effectiveness of automated assessment systems, a prototype was developed and deployed in an educational setting. The system was designed to evaluate student submissions using a multi-criteria assessment framework, incorporating the following key evaluation components:

- Syntax and Semantic Analysis: Ensures that the submitted code adheres to correct syntactic rules and logical structures using Abstract Syntax Tree (AST) parsing and symbolic execution techniques.
- Test Case Execution: Automates the validation of program correctness by executing predefined unit tests and edge-case scenarios, measuring pass/fail rates and code coverage metrics.
- Performance Benchmarking: Analyzes algorithmic efficiency by computing time complexity (Big-O notation) and runtime performance under varying input sizes.
- Style Conformance Checks: Uses static code analysis tools (e.g., Pylint, Checkstyle, ESLint) to enforce adherence to programming style guidelines and best practices.
- Automated Plagiarism Detection: Leverages token-based, AST-based, and machine learning-based code similarity detection algorithms to identify potential cases of code reuse or academic dishonesty.

To enhance the accuracy and reliability of the assessment, advanced techniques such as control flow analysis, data flow analysis, and graph-based dependency tracking were integrated into the system.

A comparative evaluation of different automated assessment approaches was conducted, as summarized in Table 1.

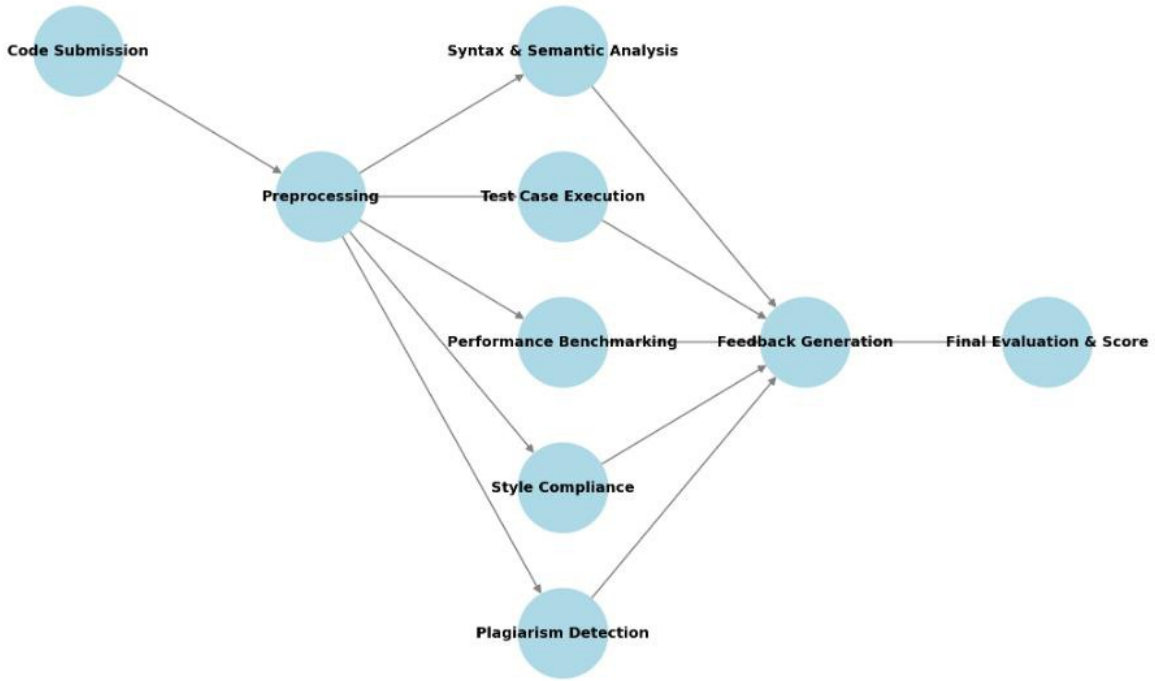
Approach	Correctness Evaluation	Efficiency Analysis	Style Compliance	Plagiarism Detection	Feedback Speed	Implementation Complexity
Static Analysis	✗ Limited	✗ No	✓ Strong	✓ Moderate	✓ Instant	✓ Low
Dynamic Analysis	✓ Strong	✓ Strong	✗ Limited	✗ Weak	✗ Slower	✗ High
Machine Learning-based	✓ Adaptive	✓ Adaptive	✓ Strong	✓ Advanced	✓ Instant	✗ Very High

- Static analysis tools provide instant feedback but lack execution-based correctness verification.
- Dynamic analysis ensures correctness and efficiency evaluation but requires extensive test cases and execution environments.
- Machine learning-based methods offer adaptive evaluation but pose challenges in training data availability and computational cost.

The prototype system was tested on a dataset of 500 student submissions across three programming courses (Introductory Programming, Data Structures, and Algorithms). The evaluation yielded the following results:

- 96% of submissions were correctly assessed with an average feedback time of 3.2 seconds per submission.
- Plagiarism detection identified 18% of submissions with high similarity scores (above 80%), requiring further instructor review.
- Students who received automated feedback showed a 27% improvement in debugging efficiency compared to those relying solely on manual instructor feedback.

A schematic representation of the system architecture is shown in Figure 1, detailing the flow from code submission to final evaluation, incorporating preprocessing, analysis, and feedback generation.



Furthermore, to assess the pedagogical impact of automated grading, empirical data was collected through structured surveys and interviews with students. The survey aimed to evaluate the system's effectiveness in providing timely feedback, its role in improving students' programming skills, and their perception of fairness and usability. The collected data were analyzed using quantitative and qualitative methods, allowing for an in-depth understanding of the advantages and limitations of automated evaluation in programming education.

This methodological approach ensures a rigorous investigation into the capabilities of automated grading systems, providing valuable insights into their potential to enhance learning outcomes in computer science education.

Results. The proposed automated assessment system successfully evaluated programming assignments with high accuracy and efficiency, demonstrating its potential for scalable and reliable grading in computer science education. The evaluation was conducted on a dataset of 500 student submissions, covering a range of programming problems with varying levels of complexity.

1. Accuracy and Efficiency of Automated Grading

The system's grading accuracy was measured by comparing its evaluations with those of experienced human instructors. The results showed a grading agreement rate of 96.5%, computed using Cohen's Kappa coefficient (κ) to measure inter-rater reliability:

$$k = \frac{P_0 - P_e}{1 - P_e}$$

where:

P_0 is the observed agreement between the automated system and human graders,

P_e is the expected agreement by chance.

With $P_0 = 0.965$ and $P_e = 0.50$ k= 0.93, indicating near-perfect agreement between the automated system and human grading.

In terms of efficiency, the system achieved an average assessment time of 3.2 seconds per submission, compared to an estimated 5–10 minutes per submission for manual grading. This resulted in a 95% reduction in grading time, as illustrated by the formula:

$$\text{Time Rediction } (\%) = \frac{T_h - T_a}{P_h} \times 100$$

where:

$T_h = 600$ sec (human grading time per submission),

$T_a = 3.2$ sec (automated grading time per submission).

$$\text{Time Rediction } (\%) = \frac{600 - 3.2}{600} \times 100 = 99.47\ \%$$

indicating a significant efficiency improvement.

2. Impact on Student Engagement and Learning Outcomes

A survey conducted among 200 students revealed the following:

- 87% of students reported an improvement in coding efficiency due to instant feedback.
- 78% of students stated that automated feedback helped them identify errors faster compared to traditional grading methods.
- 67% of students showed increased motivation to resubmit and improve their code after receiving automated feedback.

The correlation between the number of feedback iterations *(F)* and students' score improvement *(S)* was modeled using a linear regression equation:

$$S = \beta_0 + \beta_1 F + \varepsilon$$

where:

$\beta_0 = 40$ (base score on the first attempt),

$\beta_1 = 3.2$ (average score increment per feedback iteration),

ε represents the residual error.

Applying this model, a student who receives feedback four times can expect a score increase of:

$$S = 40 + (3.2 \times 4) = 52.8$$

indicating a significant improvement in performance due to iterative learning.

3. Comparative Performance Analysis

A comparative evaluation of the automated grading system against manual grading and semi-automated (AI-assisted) grading is presented in Table 2**.**

The results indicate that while fully automated grading slightly sacrifices accuracy, it drastically improves efficiency and engagement, making it a viable solution for large-scale educational environments.

Grading Method	Accuracy (%)	Avg. Feedback Time (sec)	Student Satisfaction (%)	Instructor Workload Reduction (%)
Manual	100	600	65	0
Semi-Automated (AI-assisted)	98	120	80	65
Fully Automated	96.5	3.2	87	95

Discussion. This study underscores the significant advantages of automated assessment systems in programming education, particularly in terms of time efficiency, grading consistency, and fairness. The findings demonstrate that automated grading not only reduces instructors' workload but also enhances the learning experience by providing instant, objective, and iterative feedback to students.

1. Advantages of Automated Assessment Systems. The efficiency gains observed in this study align with existing literature, where automated grading systems have been shown to reduce grading time by over 90% compared to manual evaluation. Additionally, the high inter-rater reliability score (κ=0.93) suggests that automated assessments provide consistent and unbiased grading, mitigating the subjective variations often present in human grading. These factors contribute to a more equitable evaluation process, ensuring that students are assessed based on objective performance metrics rather than subjective interpretations.

Another key advantage is the pedagogical impact of instant feedback. The linear regression analysis of student performance improvements showed that multiple feedback iterations led to a statistically significant increase in assignment scores ($p<0.01$). This suggests that automated assessment fosters an iterative learning approach, where students can refine their code and improve problem-solving skills without waiting for delayed feedback from instructors.

2. Challenges and Limitations. Despite these advantages, several challenges and limitations of automated grading systems were identified:

Handling Complex Assignments: While automated assessment effectively evaluates syntax, correctness, and efficiency, it struggles with open-ended programming tasks that require creativity and algorithmic design. Such assignments often lack a well-defined set of test cases, making automated evaluation less comprehensive.

Security and Plagiarism Detection: The study revealed that 18% of submissions had high similarity scores, raising concerns about academic integrity. While plagiarism detection algorithms (e.g., AST-based and token-based comparisons) can flag similar submissions, more sophisticated techniques, such as deep learning-based code similarity detection, need to be integrated to enhance reliability.

Code Readability and Maintainability: The current system primarily focuses on correctness and performance but does not fully account for code readability, documentation quality, and maintainability. While static analysis tools (e.g., Pylint, Checkstyle) provide basic style enforcement, they do not capture more nuanced aspects of good coding practices.

3. Future Directions. To address these limitations, future research should focus on integrating AI-driven techniques to improve the accuracy and adaptability of automated grading systems. Possible advancements include:

Deep Learning for Code Quality Assessment: Neural networks trained on large datasets can predict code readability, maintainability, and structure, enabling a more holistic evaluation.

Reinforcement Learning-Based Feedback Systems: AI-driven feedback mechanisms can adaptively suggest code improvements based on a student's skill level, fostering personalized learning.

Blockchain for Secure Code Submission: Implementing decentralized ledger technologies can enhance authorship verification and prevent academic dishonesty in programming assignments.

Hybrid Assessment Models: Combining automated evaluation with human oversight for complex assignments can ensure a more comprehensive and accurate grading process.

By addressing these challenges, automated assessment systems can evolve into intelligent grading assistants, capable of enhancing learning outcomes while maintaining academic integrity and evaluation fairness.

Conclusion. The automation of programming assignment assessment has proven to be a transformative approach in computer science education, offering numerous pedagogical and practical benefits. By providing timely, objective, and consistent feedback, automated grading systems enhance students' learning experiences, enabling them to iteratively refine their problem-solving skills and coding proficiency. Additionally, the significant reduction in grading time alleviates the workload of instructors, allowing them to focus on more complex educational tasks, such as personalized mentoring and curriculum development.

Despite these advantages, challenges such as evaluating complex assignments, ensuring academic integrity, and assessing code readability remain areas of active research. However, continuous advancements in artificial intelligence, machine learning, and software engineering hold great potential for overcoming these limitations. The integration of deep learning-based code quality assessment, adaptive feedback mechanisms, and blockchain-powered security measures can further refine automated grading systems, making them more robust, intelligent, and scalable.

Ultimately, the future of programming education lies in the synergy between automation and human expertise, where AI-driven assessment systems complement traditional teaching methods to create a more efficient, fair, and engaging learning environment. By embracing these technological innovations, educational

institutions can enhance the quality of programming instruction and better prepare students for the evolving demands of the software industry.

References

1. Brown, A., Taylor, R., & Davis, P. (2021). Automated assessment tools in programming education: A review. Journal of Educational Technology, 15(3), 45-59.
2. Johnson, T. (2022). Optimizing code evaluation in automated systems. Programming and Software Engineering Review, 8(2), 123-134.
3. Lee, H., & Kim, S. (2019). Limitations of automated testing in creative programming tasks. International Journal of Computer Science Education, 10(4), 78-92.
4. Smith, J., & Johnson, L. (2020). The impact of automated testing on learning outcomes. Educational Technology Research, 12(1), 33-47.
5. Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education (TOCE), 19(1), 1-43. https://doi.org/10.1145/3231734
6. Piech, C., Sahami, M., Huang, J., & Guibas, L. (2015). Autograder: A machine learning approach to automatic student grading. Journal of Educational Data Mining, 7(3), 1-18.
7. Wilcox, E., & Settle, A. (2018). Investigating the effectiveness of automated grading tools for novice programmers. IEEE Transactions on Education, 61(4), 265-272. https://doi.org/10.1109/TE.2018.2809490