Development of a System of Automatic Decision Evaluation in Competitive Programming Using Code Analysis Algorithms

Elmir Bekmurzaev and Zhenishbek Orozakhunov *

Posted Date: 30 April 2025

doi: 10.20944/preprints202504.2559.v1

Keywords: Artificial Intelligence; Competitive Programming; Code Analysis; OpenAI; Static Analysis; Dynamic Analysis; Automated Feedback; Programming Education; Code Quality; AI-based Evaluation System

 doi:10.20944/preprints202504.2559.v1

Article

Development of a System of Automatic Decision Evaluation in Competitive Programming Using Code Analysis Algorithms

Elmir Bekmurzaev and Zhenishbek Orozakhunov *

Ala-Too International University Bishkek, Kyrgyzstan; elmir.bekmurzaev@alatoo.edu.kg

* Correspondence: zhenishbek.orozakunov@alatoo.edu.kg

Abstract: Competitive programming is a field necessitating the accurate judging of submitted code, which is based mostly on correctness and execution time with minimal or no attention to code quality, performance in operation, and adherence to best practices. This research delves into creating an automated decision-making system for competitive programming through code analysis algorithms. By static and dynamic analysis combination, sophisticated feedback systems, and rule-based heuristics, the system is designed to deliver a stringent critique of submissions for code many times more demanding than for correctness. The system offers an improvement of feedback systems wherein by greater understanding of how they code, programmers are able to hone their craft. Literature shows that the use of sophisticated code analysis methods can greatly enhance programmers´learning trajectories as well as their competitiveness. In addition, the utilization of artificial intelligence-based methods allows for personalized feedback, thus helping individuals to improve their skills and embrace best practices in software development. This method not only enhances the learning experience but also equips programmers to handle real-world situations where code quality is as important as functionality. This paper presents the creation of a web-based system that utilizes OpenAI models to analyze and assess the programming solutions of users. The system has an in-built administrative control panel that enables organizers to create, organize, and assign custom problems to users. The uploaded code is automatically verified in terms of correctness, efficiency, and quality based on AI-driven static and dynamic analysis methods. The system provides step-by-step and personalized comprehensive feedback through which users can improve their solutions and programming skills step by step. Artificial intelligence combined with administrative flexibility positions the system as a robust teaching and assessment tool in competitive programming.

Keywords: artificial intelligence; competitive programming; code analysis; OpenAI; static analysis; dynamic analysis; automated feedback; programming education; code quality; AI-based evaluation system

I. Introduction

Competitive programming has attracted much interest among students, developers, and professionals as a means for improving problem-solving capabilities. Various competitive programming websites (e.g., Codeforces and LeetCode) utilize automatic grading systems that prioritize accuracy and speed. However, in their speed, these rating systems do not provide any feedback on essential aspects such as code maintainability, algorithm efficiency, and security measures. This lack of a comprehensive evaluation system renders programmers unaware of their shortcomings besides the general pass-or-fail requirement.

In order to fill this gap, the present study proposes a sophisticated automated decision analysis framework that is grounded in multiple analysis techniques. The platform utilizes static analysis tools for analyzing code structure, finding faults, and ensuring compliance with best practices. It also employs dynamic analysis for reviewing runtime behavior, optimizing memory usage, and

 doi:10.20944/preprints202504.2559.v1

performance measurement. By leveraging AI-based code analysis and heuristic methodologies, the platform is adept at finding poor coding practices and recommending improvements. The aim of this study is to bridge the gap between traditional evaluation methods and complete code review with the ultimate reason for providing a better learning environment for competitive programmers.

Unlike other platforms, this AI-driven evaluation system is developed using OpenAI's adaptive language models. Compared to traditional fixed-grading sites, this site employs adaptive questions that can be modified to evaluate various aspects of code quality, including security, readability, and performance. This system learns continuously because the AI models that power it are updated and upgraded on a regular basis so that it can continue to find bugs and provide feedback. Provides a detailed code review, commenting on efficiency, adherence to best practices, possible security vulnerabilities, and optimizing suggestions. With intelligent real-time feedback, the website turns competitive programming as a learning exercise into a way for users to improve not just on how quickly they program, but also on how advanced and quality the code itself is.

II. Literature Review

Competitive programming has evolved significantly from its early days, with websites like Codeforces, LeetCode, and TopCoder providing automatic code submission judging systems [8]. These systems focus most of their attention on correctness and execution time, giving little to no attention to valuable properties such as code quality, maintainability, and compliance with best practices [3]. This section reviews the existing literature on automated code judging, static and dynamic analysis, and artificial intelligence (AI) in competitive programming.

1. Automated Code Judging in Competitive Programming

Computer judges are now a central feature of competitive programming sites. As [8] explained, such judges use known test cases to adjudicate not only the correctness, but also the efficiency of the code submitted. Nevertheless, they tend not to offer detailed feedback on various dimensions of code quality, including readability, modularity, or security vulnerabilities. This restriction has been emphasized by [3], who complain that conventional systems fail to prepare programmers for the development of real-world software, in which code quality matters as much as functionality.

2. Static and Dynamic Code Analysis

Static analysis tools like the Clang Static Analyzer and SonarQube are now widely used for detecting syntax errors, code smells, and security vulnerabilities in source code. According to [5], static analysis is very good at identifying problems at an early stage in the development lifecycle. Dynamic analysis, on the other hand, examines the code while it runs, providing valuable insights into performance bottlenecks and memory consumption. Applying static and dynamic analysis together has been shown to improve code quality to a large extent, according to Chess and West (2007) [2].

3. Artificial Intelligence in Code Analysis

New developments in artificial intelligence and machine learning have opened up new possibilities for the computer-based analysis of code. Allamanis [1] discuss the application of neural networks to code summarization and bug finding, pointing to their ability to deliver more complex feedback. Similarly, Karampatsis and Sutton (2020) [6] discuss the use of AI for code pattern recognition and suggestion of improvements. These studies show that AI-based systems can be used to complement traditional analysis tools by giving deeper insights into code quality and performance.

4. Computer Programming Education Feedback Mechanisms

Constructive feedback is essential for learning in competitive programming. Hattie and Timperley (2007) [4] describe that feedback must be specific, actionable, and timely. Feedback-giving

 doi:10.20944/preprints202504.2559.v1

systems that are capable of giving overall feedback on code quality, as suggested by Keuning et al. (2017) [7], have been found to have better learning outcomes. Yet, current systems are usually marred by the inability to cater to the variety of needs for competitive programmers.

5. Integration of AI with Competitive Programming Platforms

The application of artificial intelligence in competitive programming websites is in the nascent stage. Although there are websites such as Codeforces and LeetCode that have automated judging systems, these have yet to utilize AI to carry out overall code analysis. The system discussed here tries to bridge this gap by combining static and dynamic analysis with AI-based feedback systems, as suggested by Wang et al. (2021) [9].

III. Hypothesis

The envisioned AI-based assessment platform will revolutionize competitive programming problem grading by offering intelligent, complete, and dynamic feedback compared to current binary grading.

As opposed to other current systems such as Codeforces and LeetCode that focus just on correctness and performance and neglect others such as code quality, maintainability, and learning assistance, the proposed system focuses on a holistic evaluation metric set via OpenAI adaptive models. Such models will perform nuanced, context-based feedback that improves over time and hence facilitate more effective bug checking, inefficiency checking, and deviation checking against best practices.

Instead of employing rigid test cases and fixed outcomes, the system utilizes dynamic performance criteria through editable prompts and an admin interface. This utility enables instructors or facilitators to design customized problem sets and assign assessments.

By giving importance not only to accuracy but also to aspects such as readability, efficiency, and security, the system is set to facilitate individualized feedback mechanisms whereby contestants are able to determine their weak points and enhance their skills incrementally—rendering competitive programming a systematic learning process and not a mere race against time.

In addition, having a readily usable interface for task management and feedback review will also enhance user satisfaction and adoption, particularly for students and independent learners who appreciate timely and actionable feedback on their coding practices.

A. Hypothesis: Clever Evaluation and Feedback

By incorporating OpenAI's adaptive models, the platform will be able to offer intelligent, context-sensitive feedback on code submissions, such as bug identification, inefficiencies, and best practice violations. Expected Result: The system should provide increasingly accurate, detailed, and personalized feedback with time, enhancing the quality of assessment and learning experience for the users.

B. Hypothesis: Detailed Code Analysis

Unlike other websites that focus on speed and correctness, the system evaluates code on more comprehensive factors like readability, performance, and maintainability. Anticipated Impact: The website will more impartially and holistically score and judge solutions for better coding practices and understanding beyond passing test case runs.

C. Hypothesis: Personalized Assessment Environment

It has a built-in admin interface where the educators and organizers can define user-specific tasks, set scoring criteria, and apply the scoring. Expected Result: The system should be capable of delivering a competitive learning environment with flexibility to accommodate various learning or assessment scenarios.

D. Hypothesis: Improved User Learning Experience

 doi:10.20944/preprints202504.2559.v1

Using incremental, ordered feedback, students can identify errors and enhance their submission with each new contribution, such that programming is a matter of iterative refinement under guidance. Anticipated Outcome: Users will be able to realize measurable progress over time in problem-solving ability and code quality by the follow-up evaluations. E. Hypothesis: User Experience and Satisfaction The user feedback and experience mechanisms will promote usage, particularly by students and independent learners who want immediate enhancement. Expected Result: The response and system will be user-friendly and accessible with high user satisfaction and continued use without break in learning curve.

IV. Methods

This chapter provides an overview of the assumptions involved in developing an AI-driven system for automated testing of competitive programming solutions. The assumptions are drawn from expected benefits of integrating OpenAI technologies, code analysis tools, and a bespoke task management interface.

A. Efficiency of Evaluation

Use of artificial intelligence will reduce the processing time to analyze programming entries to a great extent as compared to process-based or test-case-only processes. The machine prefers to inspect immediately with or without human intervention.

B. Intelligent Error Detection

The site would also automatically detect and classify common coding problems like syntax errors, logic errors, and performance bottlenecks. The site, based on OpenAI language models, not just detects but also provides explanations of faults.

C. Quality-Based Ranking:

The hypothesis is that, unlike traditional systems ranked on pure correctness and time of execution, this platform would be capable of generating more balanced and holistic rankings by considering other parameters such as readability, maintainability, and optimization.

D. Adaptive Evaluation:

The OpenAI feedback loop is anticipated to continue improving and refining itself, providing more accurate and useful suggestions over time as the model continues to update and retrain, resulting in improved long-term performance.

E. Flexible Task Management:

The inclusion of an administrative board will allow instructors and contest hosts to define individual issues and modify test criteria. Such flexibility will be anticipated to increase the usability of the system in diverse environments such as education, recruitment, and competition.

V. Expected Impacts

The predicted application of this system is to:

- A full web platform for competitive programming with integrated artificial intelligence-based code analysis.
- Increased learning results for the learners via detailed feedback and learning resources.
- Empirical evidence substantiates the effectiveness of the platform in evaluating and improving coding skills.
- Minimize instances of repetitive errors and promote adherence to best practices.

VI. Challenges Faced

During the development of the automatic decision analysis system, several technical and organizational challenges were encountered:

 doi:10.20944/preprints202504.2559.v1

- **AI Model Integration:** Integrating large language models into the system while maintaining fast response times and relevant output was a technically demanding task that required careful engineering and optimization.
- **Data Privacy and Security:** Ensuring the secure handling of code submissions and user data required strict attention to system architecture, including secure storage and access control mechanisms.
- **Flexible Evaluation Logic:** Designing an evaluation framework that supports a wide range of coding problems and feedback types, while keeping the backend lightweight and maintainable, proved to be a complex challenge.
- **User Interface Design:** Delivering a clean and intuitive interface for both users and administrators required multiple design iterations and ongoing feedback to ensure usability and accessibility.
- **Clarity of Feedback:** Creating feedback that is both technically accurate and understandable for beginners demanded precise tuning of AI-generated prompts and formatting to strike a balance between detail and clarity.
- **Infrastructure Constraints:** Limited computational resources imposed restrictions on the scale and speed of testing during development, which impacted model performance and iteration cycles.

Through continuous testing and iterative refinement, these challenges were gradually overcome, resulting in a more stable, user-friendly, and efficient evaluation platform.

VII. Results

The results after analyzing the performance of the platform are:

A. Administrator Control

Instructors and competition organizers will benefit from a built-in admin panel that enables the creation, editing, and scheduling of programming tasks, providing full control over problem management.

Outcome: The admin panel had control over tasks, offering full CRUD functionality (Create, Read, Update, Delete)

B. User Authentication

The website will incorporate features of login and registration, which will enable users to log into their accounts securely and monitor their history of submissions and progress over a specified period of time.

Outcome: New users are able to create account with the valid emails and login to their profiles.

C. Dynamic Assessment Criteria

The assessment system will extend beyond mere speed and accuracy; the instructors will be able to define dynamic criteria, such as items like code readability, execution time, and adherence to security policies, thereby making the platform adaptable to different learning and competition environments.

D. Support for Skill Development

With AI-based personalized feedback, not only will the students receive an idea about what they had done incorrectly but also suggestions about how to go about correcting and enhancing the code structure, organization, and quality, thereby furthering long-term skill development.

E. Feedback Mechanism

Students will receive line-by-line, context-specific explanations of their code submission, so that they can learn the rationale behind every suggestion and continue learning.

VIII. Conclusions

 doi:10.20944/preprints202504.2559.v1

Project implementation and literature review pinpoint AI-based feedback mechanisms as the key innovation for turning competitive programming from a correctness problem to an end-to-end solution for learning. Platforms like Codeforces and LeetCode, while successful in fostering algorithmic thinking, lack personalized feedback mechanisms that facilitate long-term learning of coding skills. While old-school auto-graders are scalable and effective, they fall short when dealing with more serious code quality issues such as optimization, readability, and best practices compliance. They also do not consider individual learning growth or offer actionable feedback that allows users to grow as programmers. The system therefore built bridges these loopholes by combining OpenAI- powered code review, tailored task creation through an admin interface, and personalized feedback according to configurable terms of evaluation. These features turn the system into a learning aid as well as an innovative test bed for teachers, facilitators, and learners alike. The results confirm the necessity of more intelligent and responsive approach in competitive and educational programming. Through the integration of intelligent feedback, responsiveness, and usability, the platform not only facilitates skill development but also paves the way for general use in technology education worldwide and talent evaluation.

Acknowledgment: I would like to express my heartfelt gratitude to my supervisor, M.Sc. Zhenishbek Orozakhunov, for his unfaltering support, valuable suggestions, and scholarly guidance during the period of this project. His supervision benefited me a lot to cross obstacles and remain committed to the research goals. I would like to express gratitude for the acknowledgment by the staff and faculty of the Department of Computer Science at Ala-Too International University for their encouragement towards creating a learning-friendly environment and for the feedbacks they gave during the period of this research. Lastly, I appreciate the availability of institutional resources and infrastructure that laid the groundwork for conducting this research and effectively putting the system into practice.

References

1. M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–37, 2018.
2. B. Chess and J. West, *Secure programming with static analysis*. Addison- Wesley Professional, 2007.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd ed. MIT Press, 2009.
4. J. Hattie and H. Timperley, "The power of feedback," *Rev. Educ. Res.*, vol. 77, no. 1, pp. 81–112, 2007.
5. B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *Proc. 35th Int. Conf. Software Eng. (ICSE)*, 2013, pp. 672–681.
6. R. M. Karampatsis and C. Sutton, "How often do single-statement bugs occur? The ManySStuBs4J dataset," in *Proc. 17th Int. Conf. Mining Softw. Repositories (MSR)*, 2020, pp. 573–577.
7. H. Keuning, J. Jeuring, and B. Heeren, "Towards a systematic review of automated feedback generation for programming exercises," in *Proc. 2017 ACM Conf. Innovation Technol. Comput. Sci. Educ.*, pp. 41–46, 2017.
8. S. S. Skiena and M. A. Revilla, *Programming challenges: The program- ming contest training manual*. Springer, 2003.
9. J. Wang, G. Li, and Z. Ma, "AI-driven code analysis for competitive programming: A review," *J. Comput. Sci. Technol.*, vol. 36, no. 2, pp. 245–260, 2021.
10. L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Proc. 30th AAAI Conf. Artif. Intell. (AAAI)*, 2016, pp. 1287–1293.