SUBQUIZ: VERIFYING CODE AUTHORSHIP THROUGH PERSONALIZED REFLECTIVE QUIZZES

Sukhrob Rustamovich Yangibaev

Lecturer, Department of Artificial Intelligence

Al-Khwarizmi University

Tel.: +998991352729

E-mail: s.yangibaev@akhu.uz

Madina Rustamovna Yangibaeva

Senior Lecturer, Department of Telecommunications Engineering

Urgench State University

Tel.: +998991897404

E-mail: myangibayeva03@gmail.com

Annotatsiya. SubQuiz, bu dasturlash ta'limida akademik halollikni ta'minlashga qaratilgan mualliflikni tekshirish tizimi. Tizim talabalarning kodlari asosida ularning tushunish darajasini baholash uchun har bir talaba uchun alohida tuzilgan test savollarini yaratadi. Uning samaradorligini tasdiqlash uchun nazoratli tadqiqotlar o'tkazilishi zarur.

Аннотация. SubQuiz, это система верификации авторства, предназначенная для обеспечения академической добросовестности в обучении программированию. Система генерирует персонализированные тестовые вопросы на основе студенческого кода для оценки понимания. Для подтверждения ее эффективности необходимы контролируемые исследования.

Abstract. This paper introduces SubQuiz, an authorship verification system for academic integrity in programming education. The system generates personalized quizzes from student code to assess comprehension. Controlled studies are needed to validate its effectiveness.

Keywords: academic integrity, authorship verification, LLM, personalized assessment, code comprehension, plagiarism detection, programming education

Kalit so'zlar: akademik halollik, mualliflikni tekshirish, LLM, individual baholash, kodni tushunish, plagiatni aniqlash, dasturlash ta'limi

Ключевые слова: академическая добросовестность, верификация авторства, LLM, персонализированное оценивание, понимание кода, выявление плагиата, обучение программированию

1. Introduction

The Challenge of AI-Generated Submissions. Traditional methods for ensuring academic integrity in computer science, such as MOSS [Schleimer, S., 2003: 76] or modern AI-text detectors, face new challenges in the age of LLMs [Becker, B. A., 2023: 500]. Students can use AI to generate code that includes varied naming conventions and non-optimal logic, making similarity-based detection less reliable, as structurally distinct solutions can be generated on demand. As a result, the presence of working code alone may not be sufficient evidence of a student's understanding.

The Fairness Concern. A related concern is the risk of false positives. When a student submits high-quality code, an educator might question whether AI was involved. This can create an uncomfortable situation for talented students. An objective, evidence-based method for verifying authorship could help address this concern.

Contributions of this Work. Rather than attempting to detect plagiarism through similarity analysis, SubQuiz explores a fundamental shift from detection to authorship verification, assessing whether a student truly understands the code they submitted. This paper presents SubQuiz, a system designed to:

1. Automate Reflective Assessment: Generate personalized quizzes directly from student-submitted code.
2. Verify Logic via Novel Inputs: Use a specialized prompting strategy to test understanding by changing the input data of the student's own algorithms.
3. Integrate with Existing Workflows: Provide a pipeline from Moodle submission to controlled testing and grade export.

2. Related Work

Traditional Plagiarism Detection (MOSS and JPlag). Tools like MOSS [Schleimer, S., 2003: 76] and JPlag [Prechelt, L., 2002: 1016] have long been used for detecting source code plagiarism by analyzing structural similarities between submissions. These tools were designed for human-to-human copying and remain effective in that context. However, LLMs can generate structurally distinct solutions to the same problem, which may reduce the effectiveness of similarity-based approaches for AI-generated code.

AI-Text Detectors. General-purpose AI detectors (e.g., GPTZero [Tian, E., 2023: 1], Originality.ai [Originality AI, 2023]) attempt to identify signatures of AI-generated text. When applied to source code, these tools frequently produce false positives, particularly for code written by non-native English speakers or programmers who follow strict style guides [Liang, W., 2023: 100779]. The reliability of these tools for source code remains an open question.

Manual Code Review (The Viva-Voce). The oral defense or "viva-voce," where a teacher asks a student to explain their code in real-time, remains a reliable method for verifying authorship [Sheard, J., 2002: 183]. However, it is difficult to scale. For a course with 100+ students, conducting individual interviews per assignment requires significant instructor time.

Motivation for SubQuiz. SubQuiz attempts to approximate the benefits of a manual code review in a more scalable format by automating the generation of personalized, code-reflective quizzes. It is not intended to replace existing tools but to complement them.

3. System Architecture and the SubQuiz Pipeline

The SubQuiz pipeline consists of three primary stages: (1) Submission and Contextual Analysis, (2) Personalized Quiz Generation (The Verifier AI), and (3) Controlled Execution and Integrity Monitoring.

The Verifier AI: Prompting for Understanding. The core of SubQuiz is a specialized LLM prompting strategy designed to generate questions that test understanding rather than recall. The prompt constrains the LLM with three rules:

1. Code-Reflective Variable Names: Every question must reference actual variable and class names from the student's submission.
2. The Novel Input Rule: Questions must use variable values or arguments that were not present in the student's code. For example, if a student's code creates an object with Product("Keyboard", 45.50, 20), the generated question would use entirely different values, such as Product("Mouse", 12.0, 8). This forces the student to trace their logic with unfamiliar inputs rather than recalling previously seen outputs.

3. Reflective Verification: Since the student's full code is visible on the screen during the quiz, questions avoid simple recall (e.g., "What is the name of this function?") and instead focus on logical interaction (e.g., "Given your sell method, what is the value of stock after calling sell(15) on this new object?").

Questions follow a progressive difficulty ramp: two multiple-choice questions testing simple traces and method behavior, followed by three short-answer questions requiring combined operation traces, state mutation tracking, and edge-case reasoning.

The Reflective Interface. The SubQuiz frontend is designed to mimic a code review environment. The user interface features a split-screen layout:

- Reference Panel (Left): Displays the student's full original code submission (assignmentcode). This acts as the student's "Reference Manual."
- Verification Panel (Right): Displays the personalized quiz questions. This layout ensures the assessment is "open-book," focusing strictly on the student's ability to interpret and manipulate their own code.

Figure 1. Split-screen SubQuiz interface.

This layout ensures the assessment is "open-book," focusing strictly on the student's ability to interpret and manipulate their own code. As shown in Figure 1, the split-screen design keeps the student's code visible while they answer personalized quiz questions.

Integrity and Anti-Cheat Monitoring. To maintain the rigor of a proctored exam, SubQuiz implements a robust client-side integrity layer. Key features include:

- Visibility Tracking: Detects when a student switches tabs or windows.
- Copy-Paste Blocking: Prevents students from pasting their code into external LLM windows.
- Input Debouncing and Lockouts: Automatic failure of the quiz upon repeated integrity violations.

The End-to-End Pipeline. SubQuiz is built for seamless integration into existing university workflows:

1. Import: Student code and Moodle [Dougiamas, M., 2003: 171] IDs are imported via JSON.
2. Generation: The Verifier AI processes submissions sequentially, generating a personalized question set for each student.
3. Testing: Students authenticate via their ID and take the reflective quiz in the anti-cheat controlled environment.
4. Export: Results are scaled and exported as Moodle-compatible CSV files, allowing for immediate grading integration.

4. Discussion

Expected Behavior. By design, a student who wrote their own code should be able to trace its execution with novel inputs, since they understand the logic they implemented. A student who submitted code they do not understand would need to reverse-engineer unfamiliar logic under time pressure, which we hypothesize to be significantly more difficult. However, this hypothesis has not yet been formally tested with a controlled experiment comparing known authors against non-authors.

Fairness Considerations. SubQuiz offers students an opportunity to demonstrate understanding of their submitted code through an objective quiz rather than subjective instructor judgment. A high quiz score could serve as supporting evidence of authorship. However, a low score should not be treated as definitive evidence of misconduct, as question difficulty, ambiguous wording, unfamiliarity with the quiz format, or test-taking anxiety could all contribute to low performance.

Limitations and Future Work. SubQuiz is presented as a system design; we do not report empirical evaluation data in this paper. Key areas for future work include:

- Controlled validation: Comparing quiz performance between confirmed code authors and non-authors to establish true positive/negative rates.
- Question quality assessment: Independent review of LLM-generated questions for accuracy, clarity, and appropriate difficulty.
- Ethical review: Obtaining proper consent and IRB approval before conducting studies with student participants.
- Generalizability: Testing the system across multiple courses, programming languages, and institutions.
- Confound analysis: Controlling for assignment difficulty, student fatigue, and quiz-taking conditions.

5. Conclusion

SubQuiz explores an approach to academic integrity that focuses on verifying understanding rather than detecting similarity. By generating personalized quizzes from student-submitted code and requiring students to trace their logic with novel inputs, the system provides a structured, scalable mechanism for gathering evidence of code comprehension. We believe the approach has potential, but controlled validation studies, independent question quality assessment, and proper ethical review are needed before claims about effectiveness can be made. We hope

this work contributes to the ongoing conversation about maintaining academic integrity in an era of increasingly capable AI tools.

References:

1. Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., Santos, E. A. (2023). *Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation.* – Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE). – P. 500–506.
2. Dougiamas, M., Taylor, P. (2003). *Moodle: Using Learning Communities to Create an Open Source Course Management System.* – Proceedings of the EDMEDIA Conference. – P. 171–178.
3. Liang, W., Yuksekgonul, M., Mao, Y., Wu, E., Zou, J. (2023). *GPT Detectors Are Biased Against Non-Native English Writers.* – Patterns. – Vol. 4, No. 7.
4. Originality.ai. (2023). *AI Content Detection and Plagiarism Checker*. – Available at: https://originality.ai.
5. Prechelt, L., Malpohl, G., Philippsen, M. (2002). *Finding Plagiarisms among a Set of Programs with JPlag.* – Journal of Universal Computer Science. – Vol. 8, No. 11. – P. 1016.
6. Schleimer, S., Wilkerson, D. S., Aiken, A. (2003). *Winnowing: Local Algorithms for Document Fingerprinting.* – Proceedings of the ACM SIGMOD International Conference on Management of Data. – P. 76–85.
7. Sheard, J., Dick, M., Markham, S., Macdonald, I., Walsh, M. (2002). *Cheating and Plagiarism: Perceptions and Practices of First Year IT Students.* – Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE). – P. 183–187.

8. Tian, E., Cui, A. (2023). *GPTZero: Towards Detection of AI-Generated Text.* – Available at: https://gptzero.me.