APPLYING AUTOMATIC CODE GRADING SYSTEM FOR TEACHING PROGRAMMING LANGUAGES IN UNIVERSITIES

Le Hoan*, Nguyen Quynh Anh, Pham Nhat Linh
Electric Power University

ARTICLE INFO

Received: 02/12/2023
Revised: 27/12/2023
Published: 27/12/2023

KEYWORDS

AutoChecking
Learning management systems
Automatic Code Grading System
Automated assessment tools
Sandbox environment

ABSTRACT

The rapid growth of online education platforms highlights the need for effective automatic grading systems, especially for programming courses in universities. This paper introduces an Automatic Code Grading System named AutoChecking, tailored for evaluating code submissions in various programming languages at the university level. AutoChecking combines static code analysis and dynamic execution to ensure thorough and equitable assessment of students' programming assignments. It features a sandbox environment for securely running student codes against predefined test cases, evaluating not just accuracy but also efficiency and coding style. The system is adaptable, with a flexible scoring mechanism to meet different course and instructor needs. AutoChecking integrates with existing online Learning Management Systems to streamline submission, grading, and feedback, making it easier for instructors to manage assignments and monitor student progress with detailed analytics. We evaluated the effectiveness of the AutoChecking through a semester-long deployment in introductory and advanced programming courses at three universities. The results showed that AutoChecking saves instructors' time, provides consistent feedback to students, enhancing their learning, and plays a crucial role in maintaining academic integrity by detecting code similarities.

ÁP DỤNG HỆ THỐNG CHẤM ĐIỂM TỰ ĐỘNG CHO VIỆC GIẢNG DẠY NGÔN NGỮ LẬP TRÌNH TRONG CÁC TRƯỜNG ĐẠI HỌC

Lê Hoàn*, Nguyễn Quỳnh Anh, Phạm Nhất Linh
Trường Đại học Điện Lực

THÔNG TIN BÀI BÁO

Ngày nhận bài: 02/12/2023
Ngày hoàn thiện: 27/12/2023
Ngày đăng: 27/12/2023

TỪ KHÓA

AutoChecking
Hệ thống học tập trực tuyến
Hệ thống chấm điểm tự động
Công cụ đánh giá tự động
Môi trườngSandbox

TÓM TẮT

Sự phát triển nhanh chóng của các nền tảng giáo dục trực tuyến đã làm nổi bật nhu cầu về hệ thống chấm điểm tự động hiệu quả, đặc biệt là cho các khóa học lập trình tại các trường đại học. Bài báo này giới thiệu Hệ thống chấm điểm tự động được gọi là AutoChecking, được thiết kế riêng cho việc đánh giá các bài nộp mã chương trình bằng nhiều ngôn ngữ lập trình ở cấp đại học. AutoChecking kết hợp phân tích mã code tĩnh và thực thi động để đảm bảo việc đánh giá kỹ lưỡng và công bằng các bài tập lập trình của sinh viên. Hệ thống bao gồm một môi trường sandbox an toàn để chạy mã của sinh viên chống lại các trường hợp kiểm thử được định sẵn, đánh giá không chỉ độ chính xác mà còn hiệu suất và phong cách lập trình. Hệ thống có khả năng thích ứng với cơ chế tính điểm linh hoạt để đáp ứng nhu cầu của các khóa học và giảng viên khác nhau. AutoChecking tích hợp với các Hệ thống học tập trực tuyến hiện có để đơn giản hóa quy trình nộp bài, chấm điểm và phản hồi, giúp giảng viên dễ dàng quản lý bài tập và theo dõi tiến độ của sinh viên thông qua phân tích chi tiết. Chúng tôi nghiên cứu và triển khai AutoChecking qua một học kỳ tại ba trường đại học, kết quả cho thấy hệ thống này không chỉ tiết kiệm thời gian cho giáo viên mà còn cung cấp phản hồi nhất quán cho sinh viên, nâng cao trải nghiệm học tập và thúc đẩy tính chính trực học thuật bằng cách phát hiện sự tương đồng trong các bài nộp mã chương trình.

DOI: https://doi.org/10.34238/tnu-jst.9328

* Corresponding author. *Email: hoanle@epu.edu.vn*

1. Introduction

In the context of digital transformation altering educational methodologies, university computer science departments globally are integrating advanced technologies to meet contemporary pedagogical needs. The growing enrollment in computer science, fueled by the market demand for skilled programmers, has led to the need for scalable, efficient assessment tools. Automatic Code Grading Systems (ACGS) have emerged as a key solution, innovatively addressing the intensive demands of large-scale programming education [1].

The advent of the ACGS has transformed the educational landscape by offering immediate, detailed feedback on code assignments, overcoming previous limitations due to time constraints on instructors [2]. The ACGS utilize a blend of computational methods, including static code analysis and dynamic code execution, to thoroughly evaluate student submissions [3]. Static analysis checks coding standards and identifies anti-patterns, while dynamic execution tests code functionality against predefined test cases, ensuring compliance with functional requirements.

The rise of academic dishonesty, especially code plagiarism, is a significant issue in academia, exacerbated by easy access to online code resources. To address this, the ACGS have incorporated advanced plagiarism detection algorithms, utilizing fingerprinting, string matching, and machine learning to identify similarities in code submissions [4]. These tools are essential in upholding academic integrity and ensuring the distinct assessment of each student's work.

The integration of the ACGS with existing the LMS system is crucial for their effective implementation in universities. This integration streamlines the workflow of assignment submission, grading, and feedback within the familiar LMS environment, thereby reducing resistance to new technology adoption and enhancing acceptance among faculty and students [5].

The ACGS have emerged as a key innovation in computer science education, addressing scalability, instant feedback, and uniform assessment challenges in programming courses. Initially, ACGS research centered on basic syntax and logic checks [6]. However, advancements in these systems have shifted focus towards more advanced tools providing personalized feedback and accommodating various programming paradigms [5]. Additionally, integrating ACGS with other educational technologies, particularly for smooth the LMS system compatibility, has become an essential research focus [2].

The accuracy of code evaluation is a critical aspect in developing the ACGS. A comparative study highlighted various approaches like test-based grading, static, and dynamic analysis, showing their effectiveness varies by context. Nayak et al. [7] suggest that while test-based methods are common, integrating static and dynamic analysis can improve grading robustness. However, their study didn't address the learner's capability to verify code validity and success or failure outcomes.

From the perspective of user experience, recent studies have examined the acceptability and usability of ACGS among students and instructors. Figueira et al. [8] specifically addressed the need for systems that are intuitive and minimally disruptive to the existing pedagogical flow. Moreover, feedback from users indicates that the immediate response provided by ACGS can significantly enhance the learning process [9].

The scalability of ACGS is crucial for their deployment in university settings. The research by Kleij et al. [10] analyzed the performance of these systems in massive open online courses (MOOCs) and large in-person classes, demonstrating how ACGS can cope with a high volume of submissions without compromising grading consistency. In the context of large-scale classes, the efficiency and scalability of ACGS are paramount. Research by Vu et al. [11] highlighted the potential for ACGS to manage the increased student load while maintaining high standards of assessment. The ability of these systems to quickly adapt to various course sizes and complexity levels has become a crucial aspect of their design [1]. The integration of ACGS with LMS system and other educational tools is vital for creating a cohesive learning environment. Panjaitan et al.

[12] explored the interoperability of ACGS with various LMS platforms, revealing the challenges and opportunities associated with data exchange and system communication.

The research on the automatic grading system, AutoChecking, focuses on features like grading student-written functions or programs based on teacher requirements, providing immediate results, identifying syntax errors, and indicating datasets that yield correct or incorrect outputs. Additional configurable options include allowing students to review submissions, make single or multiple submissions, manage program runtime, and implement different scoring systems. The study emphasizes the impact of these factors on student outcomes and response times. The key objectives are to analyze and evaluate students' learning outcomes using AutoChecking and its integration with LMS system, aiming to improve students' fundamental understanding and programming skills.

2. Research methods

2.1. AutoChecking

Based on the highlights features of the ACGS system and the values, it brings in modern education such as *automated code evaluation, immediate feedback, scalability, language support, intergration with LMS, accessibility and convenience, ect.* AutoChecking is researched and developed as a plugin that can be intergrated with LMS, exemplifies this relationship by serving as a specialized question type that allows teachers to automate the grading of code written by students. Essentially, AutoChecking functions within the broader ecosystem of ACGS, augmenting their capabilities by providing a seamless, integrated experience within LMS. On the one hand, in order to efficiently evaluate and grade programming assignments. ACGS are designed to handle a wide range of programming languages and tasks, assessing code for correctness, efficiency, and adherence to specific requirements. AutoChecking extends this functionality within LMS's framework, enabling teachers to create custom coding questions and automatically run student-submitted code against pre-defined test cases. This integration streamlines the grading process, reducing the manual effort required by instructors and providing immediate, objective feedback to students. On the other hand, AutoChecking’s integration into LMS enhances the learning experience by allowing for a cohesive, all-encompassing educational platform. Students can write, submit, and receive feedback on their code all within the same system where they access course materials and other resources. This integration represents a significant step forward in educational technology, demonstrating how AutoChecking can revolutionize the way programming is taught and assessed in an online learning environment.

AutoChecking, a question format within LMS, allows educators to evaluate student responses programmatically in programming and other computer science and engineering courses. It is primarily used for grading code-based assignments, where student work is assessed through a series of test runs. In quizzes, AutoChecking features an adaptive 'Check' button for students to verify their code against specified criteria, with options for resubmission often involving minor penalties. Grading is typically based on passing all tests or the number of successful test cases. Additionally, it offers a flexible penalty system, ranging from no penalty for resubmissions to incremental penalties for further incorrect attempts, or even strict penalties for any incorrect submission.

AutoChecking supports text-based programming languages, including C, C++, Python, and can handle various question complexities, from simple code completion to larger assignments. As a question type in LMS like Moodle [13], it integrates with automated question formats (multiple-choice, numerical, short answer, matching) and manually graded essays. It's beneficial in lab learning environments, incorporating educational content into quizzes, like tutorials on if-statements or loops. Being web-based, it offers flexibility for students to complete tasks remotely, reducing reliance on fixed lab sessions, which can be repurposed for additional support.

However, to evaluate the effectiveness and impacts of AutoChecking as an educational tool on learning programming, we define a number of criteria such as *experimental studies, comparative studies, qualitative analysis, quantitative analysis, case studies, longitudinal studies*.

1. *Experimental studies*: Conducting research with actual students and educators using AutoChecking in real-world classroom settings. This approach can provide valuable data on how AutoChecking affects learning outcomes, engagement, and comprehension of programming concepts.
2. *Comparative studies:* Comparing the results of traditional code assessment methods with those obtained using AutoChecking. This could involve looking at variables like student performance, time efficiency, and satisfaction.
3. *Qualitative analysis:* Gathering qualitative data through interviews, focus groups, and surveys from students and educators. This data can provide insights into the usability, perceived benefits, and potential drawbacks of AutoChecking.
4. *Quantitative analysis:* Employing statistical methods to analyze data gathered from AutoChecking's use, such as grades, completion rates, and the frequency of specific types of errors made by students.
5. *Case studies:* Detailed examination of specific instances where AutoChecking has been implemented, to understand its impact in various educational contexts and settings.
6. *Longitudinal studies:* Observing and analyzing the effects of AutoChecking over an extended period to assess its long-term impact on learning and teaching programming.

These research methods can help educators, developers, and researchers understand the effectiveness of AutoChecking in an educational setting and contribute to its ongoing improvement and adaptation to various learning environments.

2.2. The Architecture of AutoChecking

This section outlines the development of an AutoChecking plugin for Learning Management Systems (LMS), designed to automate code grading. As a question-type plugin, it enables educators to grade student responses programmatically. AutoChecking can execute student-submitted code across various programming languages, and will be used in LMS's adaptive quiz mode. Educators set task requirements and test datasets, while learners complete tasks and submit answers, receiving immediate results upon submission. A block diagram, shown in Figure 1, details AutoChecking's components and the sequence of activities from test submission to result reception.

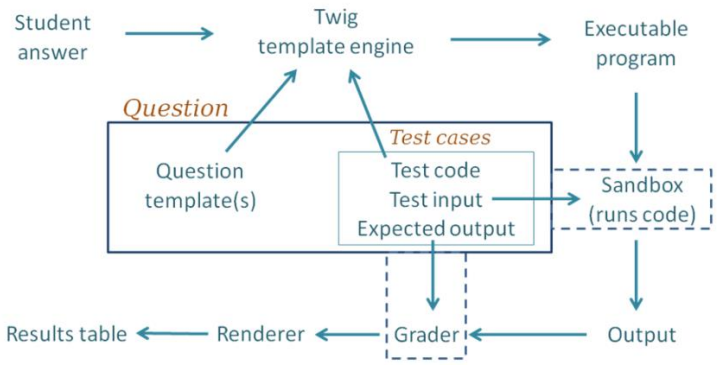

Figure 1. *Architecture of AutoChecking*

Following with each stage of the grading process sequentially:

1. For every test case, the Twig template engine. Twig is a flexible, fast, and secure template engine for PHP. In the context of AutoChecking, a question-type plugin for LMS, Twig can be

used to generate questions and test cases dynamically. Here's a general overview of how Twig might be integrated into AutoChecking for this purpose. Twig, in the context of AutoChecking, enhances the capabilities of the AutoChecking plugin by allowing for dynamic and flexible question generation. This integration elevates the potential of LMS's assessment tools, making them more adaptable, engaging, and effective for students and instructors alike in the realm of programming education. It combines the student's answer with the question's template and includes code specific to that test case. This combination produces a program that is capable of execution. When we say "executable," we refer to a program that can be run, potentially requiring an initial compilation step.

2. The Jobe sandbox receives the executable program, where it proceeds to compile the program if needed and executes it. It utilizes the standard input provided by the test case during this execution.

3. The output generated during the program's execution is directed to the designated Grader component, along with the anticipated output specified for the test case. While the "exact match" grader is prevalent, various other grader types are also accessible.

4. The grader produces a "test result object" encompassing, among other elements, attributes labeled as "Expected" and "Got."

5. The described process is iterated for every test case, resulting in an array of test result objects (not explicitly displayed in the illustration).

6. The AutoChecking question renderer receives all the test outcomes and displays them to the user in a Results Table format, as shown in Figure 2. Successful tests are denoted by a green checkmark, while unsuccessful ones are marked with a red cross. Usually, the entire table is colored red if any tests fail or green if all tests succeed.

	Input	Expected	Got	
✔	Monty Python 0 4	Mont	Mont	✔
✔	Sinh vien Su pham 10 17	Su pham	Su pham	✔
✔	He quan tri co so du lieu 3 11	quan tri	quan tri	✔
✔	Nho lam bai tap day du, khong se bi diem thap 0 20	Nho lam bai tap day	Nho lam bai tap day	✔
✔	Training Code khoang 1-2 gio moi ngay 13 37	khoang 1-2 gio moi ngay	khoang 1-2 gio moi ngay	✔

Figure 2. *Results table after successful tests of students*

2.3. Question types in AutoChecking

AutoChecking supports a wide array of question formats, with the flexibility to incorporate more. The definition of an AutoChecking question type relies on a prototype that specifies execution parameters like programming language and environment, and a blueprint for constructing test programs from test cases and student input. It dictates evaluation methods, including *EqualityGrader*, *NearEqualityGrader*, *RegexGrader*, or *TemplateGrader*, to assess submission accuracy.

- ***EqualityGrader:*** requires an exact correspondence between the student's output and the pre-defined correct output for the testcase.

- ***NearEqualityGrader***: operates similarly, but it is not sensitive to case differences and allows for minor discrepancies in whitespace, such as absent or extra blank lines, or spaces where fewer are anticipated.

- ***RegexGrader:*** interprets the Expected output as a regular expression (excluding PERL-style delimiters) and searches for a match anywhere within the output. For instance, an expected pattern of 'ab.*z' would align with any output that includes 'ab' followed by any characters and then a 'z' further on. To match the entire output, the regular expression should begin with '\A' and conclude with '\Z'. The matching process employs MULTILINE and DOTALL settings.

- ***TemplateGraders***: offer complexity but also provide question authors with extensive control over the execution, evaluation, and presentation of results; this is further elaborated in the section concerning Grading with templates.

The *EqualityGrader* is favored for standard applications, encouraging students to achieve exact accuracy in their outputs. It allows resubmissions with minor penalties, often more effective than allocating partial marks via regular expression matches. Test cases, designed by the question creator, evaluate student code using test code, standard input, and expected output. Authors can also add extra files to the execution context. The system uses templates to merge student submissions with test data, creating unique test programs for each case or combining multiple cases in a single execution. More information on templates is available in the Templates section.

2.4. A template question in AutoChecking

The C-function question type requires students to provide a C function, along with any necessary auxiliary functions, according to a given specification. For a simple illustration, the question might be, "Compose a C function with the signature int sqr(int n) that computes and returns the square of its argument n." The creator of the question would then supply some test scenarios, for example in Figure 3.

```
printf("%d\n", sqr(-11));
```

Figure 3. *The question for a test scenario*

And present the expected output for this test. A per-test template for this kind of question would encapsulate both the student's submission and the test code into a unified program, as in Figure 4.

```
#include <stdio.h>
// --- Student's answer is inserted here ----
int main()
{
   printf("%d\n", sqr(-11));
   return 0;
}
```

Figure 4. *The per-test template*

The program is compiled and executed for each test case, with its output compared against a predefined expected output (121) to determine correctness. This approach uses a per-test template, contrasting with the complex combinator template of pre-configured C function questions. For detailed information, see the section on Templates.

2.5. Intergrate AutoChecking into the LMS system

LMS systems like Moodle, Canvas, and Blackboard are central to modern education, offering a digital framework for managing and tracking educational activities. They provide a unified platform for educators and learners to interact, share resources, and streamline learning processes, catering to a variety of educational needs across diverse settings. To take advantage of the advantages of LMS, we use the online learning system Moodle to integrate the AutoChecking module into the LMS [14]. Moodle, an acronym for "Modular Object-Oriented Dynamic Learning Environment," stands as a prominent open-source LMS, renowned for its versatility and

adaptability in educational settings worldwide. Created by Martin Dougiamas in the early 2000s, Moodle has evolved into a comprehensive platform that caters to diverse learning needs across various institutions, from schools to universities and corporate entities. Figure 5 illustrates its role in transforming the learning experience when intergrating with LMS.

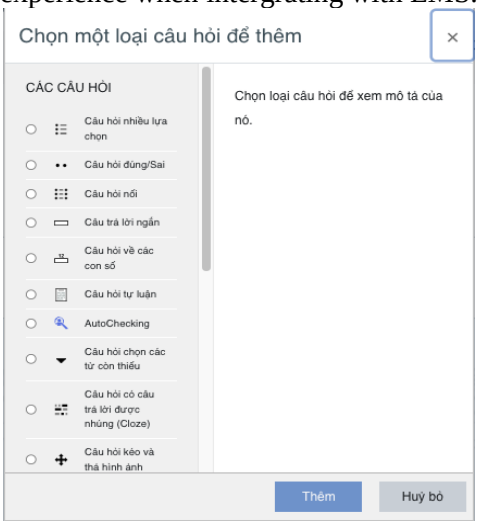

Figure 5. *The AutoChecking module in LMS*

This integration offers multiple benefits: it lightens educators' grading workload, allowing more time for personalized feedback, and ensures consistent, fair evaluation by uniformly applying grading criteria to all submissions.

AutoChecking provides students with instant feedback, helping them identify strengths, weaknesses, and areas for improvement. This fosters iterative learning and self-assessment in coding skills. Its integration with modern educational demands and technology prepares students for technology-centric careers. The fusion of AutoChecking with the LMS system signifies a progressive approach, enhancing interaction and efficiency in the learning environment. The interface of the AutoChecking module after integration will be interacted with students. Figure 6 shows what the student sees after a correct submission. Figure 7 shows the state after an incorrect submission.

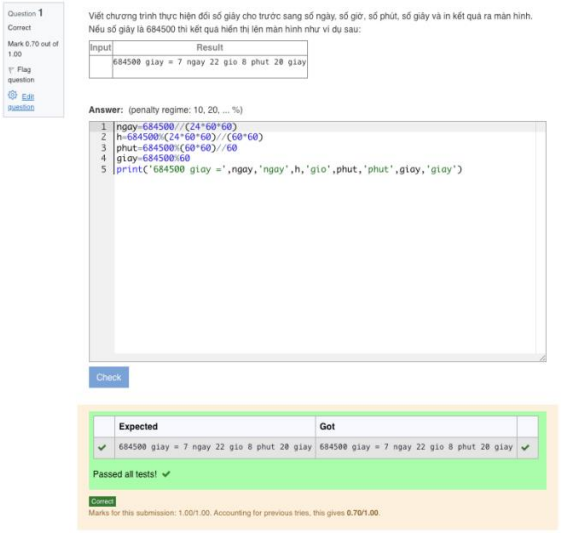

Figure 6. *The simple Python question, correctly answered*

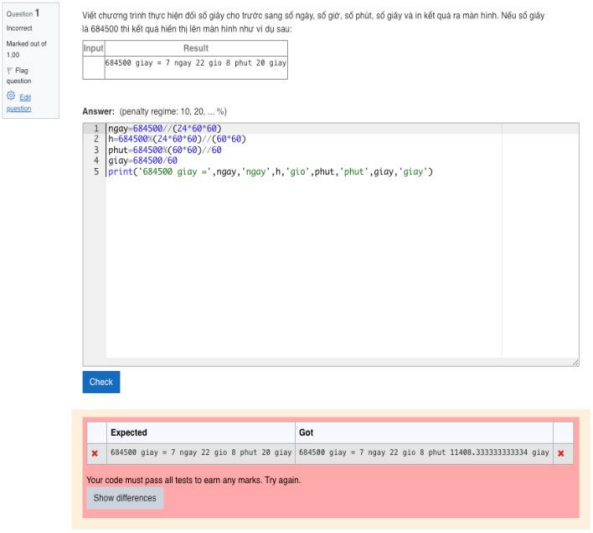

Figure 7. *The simple Python question, wrongly answered*

Here are some essential aspects to be aware of:

- Students can instantly see their feedback after they hit the Check button.
- This feedback is represented in a tabular format listing the tests conducted, the expected results for each test from the student's function, and the actual results obtained. Correct answers are marked with green checks, while incorrect ones are signified by red crosses.
- If any of the outputs are incorrect, the feedback section turns red, a reflection of the "all or nothing" grading approach, resulting in no marks being assigned. Upon the student's submission of a correct response (as shown in Figure 6), the entire feedback section changes to green, and the student receives a full score, reduced by any accumulated penalties, which, in this instance, amounts to 10%.

3. Results and Discussions

In order to demonstrate an experimental testbed, we research and test AutoChecking based on an open source module. It is an open-source, no-cost plugin of question-type for LMS, capable of executing program code provided by students in response to a diverse array of programming queries across multiple languages. Primarily designed for application in computer programming education, its utility extends to the grading of any textual response-based questions. Additionally, for users with advanced expertise, it offers the capability to formulate questions in HTML, supplemented by JavaScript. This feature significantly broadens the scope of questions that can be effectively handled within the system. We evaluate the effectiveness and feasibility of the automated code grading module integrated into the LMS during the teaching process, we conducted a trial with 30 students of equal academic ability from a Python Programming course, by dividing 30 students into 2 groups (each group consisting of 15 students), as follows:

Group 1: Students complete programming assignments on the Quiz system that the teacher has created on LMS system, submit their work themselves, and the submissions are graded using the automated code grading module developed on LMS (students can submit their work multiple times for self-assessment of the results).

Group 2: Students complete programming assignments and the teacher grades them using traditional methods.

With the two groups of students, we utilized the designed exercise system and conducted experiments in teaching the Python programming language through five sessions focusing on the topic of List data types. Ultimately, we carried out evaluations through two tests for each group as follows:

- ***Group 1 (Quiz):*** Complete quizzes created by the teacher on LMS system, submit their work themselves, and the submissions are graded using the automated code grading module developed on the LMS system.
- ***Group 2 (Traditional):*** Take the test and have the teacher grade them using traditional methods (grading by testing each student's work individually).

The tests for both groups were identical in content, consisting of 8 programming assignments to be completed within 120 minutes, corresponding to 8 questions from Q1 to Q8.

From the above test, we obtained the following results:

- ***In terms of scores***: The results of Group 1's work are shown in Table 1, and the results of Group 2's work are shown in Table 2.
- ***In terms of grading time***: Group 1: The time taken to grade assignments depends on the run time of a test for one assignment. On average, the time to run one test is 1000ms. With 15 students taking the test, and each test consisting of 8 assignments with 10 tests each, the total grading time for Group 1 is: 15 * 8 * 10 * 1000ms = 1200000ms = 1200 seconds. Group 2: The average grading time per student in Group 2 is 5 minutes per assignment. With 15 students, the total grading time for Group 2 is 5 * 8 * 15 = 600 minutes.

From the results, it is clear that the grading time for Group 1 is much faster compared to Group 2. Based on these results, it can be observed that teaching using the LMS system integrated with the AutoChecking automated code grading module has initially been effective in enhancing student learning in the Python programming language course.

Table 1. *Test results of Group 1*

Fullname	Q1/1.25	Q2/1.25	Q3/1.25	Q4/1.25	Q5/1.25	Q6/1.25	Q7/1.25	Q8/1.25	Score/10
ĐỖ ĐĂNG TÙNG	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25	**10.00**
VŨ ĐỨC DUY	0.00	0.00	0.00	0.83	1.25	1.25	1.25	1.25	**5.83**
VÕ BÌNH THẮNG	0.88	1.00	1.13	1.25	1.25	1.25	1.25	0.00	**8.01**
NGUYỄN ĐỨC TUẤN	1.25	0.63	1.25	1.25	1.25	1.25	1.25	0.42	**8.55**
TRƯƠNG QUỐC VINH	1.25	1.25	1.25	1.25	1.25	1.25	1.25	0.42	**9.17**
TRẦN VĂN HOÀN	0.88	1.25	1.25	1.25	1.25	1.25	0.83	0.00	**7.96**
TRẦN THU HÀ	1.25	1.25	1.25	1.25	1.25	0.00	1.25	1.25	**8.75**
TRẦN MINH ANH	0.00	0.00	1.00	1.25	1.25	1.25	1.25	0.00	**6.00**
TRẦN HỮU CHÂU MINH	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25	**10.00**
TẠ QUANG ĐẠT	1.00	1.25	1.25	1.25	0.42	1.25	0.00	1.25	**7.67**
NGUYỄN QUANG HÀ	0.00	0.00	0.00	1.25	0.00	1.25	0.00	1.25	**3.75**
PHAN VĂN PHONG	0.75	1.25	1.25	0.00	0.00	1.25	1.25	1.25	**7.00**
PHẠM TRUNG HIẾU	1.13	1.25	1.00	1.25	1.25	0.00	1.25	1.25	**8.38**
PHÀM THÀNH LONG	0.00	1.00	0.00	0.00	1.25	1.25	1.25	0.00	**4.75**
PHẠM DUY ĐỨC	1.00	0.00	0.88	1.25	1.25	1.25	0.42	1.25	**7.30**

Table 2. *Test results of Group 2*

Fullname	Q1/1.25	Q2/1.25	Q3/1.25	Q4/1.25	Q5/1.25	Q6/1.25	Q7/1.25	Q8/1.25	Score/10
NGUYỄN HẢI ANH	1.00	0.90	1.20	0.40	0.90	0.00	0.30	1.00	**5.70**
ĐÀO THỊ CHANG	0.20	0.10	0.50	0.00	0.20	0.00	0.20	1.00	**2.20**
NGUYỄN THẾ ĐẠI	1.20	0.10	0.90	0.60	1.00	1.20	0.90	0.10	**6.00**
LÊ MINH ĐẠT	0.90	0.30	0.10	0.30	0.30	0.80	0.70	0.80	**4.20**
NGUYỄN THÀNH ĐẠT	0.00	0.20	1.00	0.30	0.80	0.50	0.80	1.20	**4.80**
ĐỖ THỊ DIỄM	0.50	0.80	0.50	0.80	1.00	0.20	1.00	0.30	**5.10**
DƯƠNG TIẾN ĐỨC	0.40	1.20	0.50	1.20	0.30	1.10	0.50	0.70	**5.90**
PHAN THỊ THÙY DUNG	0.90	0.90	0.50	1.20	0.00	1.00	0.90	0.10	**5.50**
LÊ TUẤN DŨNG	0.70	0.00	1.20	0.30	0.10	0.50	1.20	0.40	**4.40**
LƯỜNG TIẾN DŨNG	0.00	0.30	1.20	0.50	0.60	0.20	0.60	0.40	**3.80**
NGUYỄN TRỌNG DŨNG	0.50	0.40	1.00	1.20	0.80	0.40	1.20	1.10	**6.60**
TRẦN TUẤN DŨNG	0.10	0.40	1.20	0.80	0.30	0.80	0.30	0.80	**4.70**
TRƯƠNG VĂN DƯƠNG	0.70	1.10	1.00	1.00	0.80	0.60	0.60	0.40	**6.20**
CHỬ TRUNG HIẾU	0.90	0.30	0.90	1.00	0.40	0.30	0.80	0.90	**5.50**
LÊ TRUNG HIẾU	0.60	0.40	0.50	0.60	0.30	1.00	0.70	0.60	**4.70**

Based on experiments, evaluations and comparisons of actual results when deployed at three universities, AutoChecking has solved the criteria we outlined in section 2.1, including *experimental studies, comparative studies, qualitative analysis, quantitative analysis, case studies, longitudinal studies.*

4. Conclusions

The introduction of the ACGS, AutoChecking, represents a significant advancement in educational technology, particularly beneficial for programming and computer science education.

When integrated with the LMS system, it offers notable pedagogical and administrative advantages. It facilitates active learning by providing immediate feedback on code submissions, aiding students in quickly recognizing and correcting their mistakes, thereby deepening their programming understanding. Its ability to handle various programming languages and question types showcases its flexibility for diverse curricula and teaching methods, applicable across different learning levels. The integration of AutoChecking into the LMS architecture has been practically tested with 30 students in a Python programming course, yielding positive results that highlight its real-world applicability. Future plans include expanding its use to other programming languages like PHP, Java, C#, and enhancing its capabilities to detect and prevent plagiarism in student submissions.

Acknowledgements

The author would like to thank the referees for giving precious comments and suggestions. This work was done under the partial support of the Science and Technology Foudation of Electric Power University under grant number ĐTKHCN.24/2022.

REFERENCES

[1] J. C. Paiva, J. P. Leal, and Á. Figueira, "Automated Assessment in Computer Science Education: A State-of-the-Art Review," *ACM Trans. Comput. Educ.*, vol. 22, no. 3, pp. 34:1-34:40, Jun. 2022, doi: 10.1145/3513140.

[2] J. Schneider, R. Richner, and M. Riser, "Towards Trustworthy AutoGrading of Short, Multi-lingual, Multi-type Answers," *Int. J. Artif. Intell. Educ.*, vol. 33, no. 1, pp. 88–118, Mar. 2023, doi: 10.1007/s40593-022-00289-z.

[3] Z. Cakmak and I. Akgün, "A Theoretical Perspective on the Case Study Method," *J. Educ. Learn.*, vol. 7, p. 96, Oct. 2017, doi: 10.5539/jel.v7n1p96.

[4] V. Ljubovic and E. Pajic, "Plagiarism Detection in Computer Programming Using Feature Extraction From Ultra-Fine-Grained Repositories," *IEEE Access*, vol. 8, pp. 96505–96514, 2020, doi: 10.1109/ACCESS.2020.2996146.

[5] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Machine Learning-Based Automated Grading and Feedback Tools for Programming: A Meta-Analysis," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, in ITiCSE 2023. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 491–497, doi: 10.1145/3587102.3588822.

[6] N. G. Fonseca, L. Macedo, M. J. Marcelino, and A. J. Mendes, "Augmenting the teacher's perspective on programming student's performance via permanent monitoring," in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct. 2018, pp. 1–9, doi: 10.1109/FIE.2018.8658924.

[7] S. Nayak, R. Agarwal, and S. K. Khatri, "Automated Assessment Tools for grading of programming Assignments: A review," in *2022 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2022, pp. 1–4, doi: 10.1109/ICCCI54379.2022.9740769.

[8] Á. Figueira, J. P. Leal, and J. C. Paiva, "Automated Assessment in Computer Science Education: A State-of-the-Art Review," 2022. [Online]. Available: https://repositorio.inesctec.pt/handle/123456789/14301. [Accessed Nov. 29, 2023].

[9] S. Marwan, G. Gao, S. Fisk, T. W. Price, and T. Barnes, "Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science," in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 194–203, doi: 10.1145/3372782.3406264.

[10] F. M. Van der Kleij, R. C. W. Feskens, and T. J. H. M. Eggen, "Effects of Feedback in a Computer-Based Learning Environment on Students' Learning Outcomes: A Meta-Analysis," *Rev. Educ. Res.*, vol. 85, no. 4, pp. 475–511, Dec. 2015, doi: 10.3102/0034654314564881.

[11] L. Vu, "A Case Study of Peer Assessment in a Composition MOOC: Students' Perceptions and Peer-Grading Scores vs. Instructor-Grading Scores," in *Learning and Performance Assessment: Concepts, Methodologies, Tools, and Applications*, IGI Global, 2020, pp. 918–956, doi: 10.4018/978-1-7998-0420-8.ch043.

[12]B. M. Panjaitan, S. A. Rukmono, and R. S. Perdana, "Integration Model for Learning Management Systems, Source Control Management, and Autograders using Service-Oriented Architecture Principle," in *2021 International Conference on Data and Software Engineering (ICoDSE)*, Nov. 2021, pp. 1–6, doi: 10.1109/ICoDSE53690.2021.9648450.

[13]"Home | Moodle.org." [Online]. Available: https://moodle.org/ [Accessed Nov. 28, 2023].

[14]"Never Stop Learning - IT for all." [Online]. Available: https://courses.it4all.vn/ [Accessed Dec. 04, 2023].