Improving Coding Assignments with Partial Auto-Grading and Immediate Feedback: Report from an intervention

Improving Coding Assignments with Partial Auto-Grading and Immediate Feedback: Report from an intervention

Marco L. Della Vedova
marco.dellavedova@chalmers.se

September 8, 2025

Abstract

This paper presents a case report on the use of partial auto-grading with immediate feedback in the course *Introduction to Artificial Intelligence* (MMS131) at Chalmers University of Technology. The intervention aimed to reduce grading workload while providing students with timely, formative feedback on coding assignments. Evaluation was carried out by comparing the 2023 course iteration (manual grading) with the 2024 iteration (auto-grading integrated into Jupyter notebooks). The case showed reduced grading time, shorter feedback delays, and lower resubmission rates, alongside generally positive perceptions from both students and teaching staff. Challenges included installation issues and occasional concerns about the clarity of automated tests. We discuss the pedagogical and operational implications of these findings and reflect on design choices such as scaffolding with a preliminary assignment and balancing visible and hidden tests.

Abstract

Denna artikel presenterar en fallstudie om användningen av partiell auto-rättning med omedelbar återkoppling i kursen *Introduction to Artificial Intelligence* (MMS131) vid Chalmers tekniska högskola. Interventionen syftade till att minska arbetsbördan för rättning samtidigt som studenterna fick snabb, formativ återkoppling på sina programmeringsuppgifter. Utvärderingen genomfördes genom att jämföra kursupplagan 2023 (manuell rättning) med upplagan 2024 (auto-rättning integrerad i Jupyter notebooks). Fallstudien visade på minskad rättningstid, kortare väntetider för återkoppling och färre om-inlämningar, tillsammans med överlag positiva upplevelser från både studenter och lärare. Utmaningar inkluderade installationsproblem och viss oro kring tydligheten i de automatiska testerna. Vi diskuterar de pedagogiska och praktiska implikationerna av resultaten och reflekterar kring designval såsom användning av en inledande uppgift för att underlätta introduktion samt balansen mellan synliga och dolda tester.

Keywords: *coding; assignments; auto-grading; immediate feedback*

1 Introduction

In the course Introduction to Artificial Intelligence (MMS131) at Chalmers University of Technology, students are required to complete assignments that combine programming tasks with written, pencil-and-paper exercises. The grading process for these assignments was particularly critical and labor-intensive due to two main factors: (i) The course consistently enrolls a large number of students, i.e., about 200; (ii) All assignments were manually graded by teaching assistants (TAs) and lecturers.

This traditional grading approach presents challenges that affect both teaching and learning:

- **Delayed feedback**: Manual grading of all exercises for all students leads to significant delays in returning grades and feedback.
- **High workload**: The grading burden on teaching staff is substantial, reducing the time available for other pedagogical activities.

To address these challenges, this work explored introducing an auto-grading system for the programming exercises. This system is designed to provide immediate, automated feedback to students, thereby enabling them to identify and correct errors in a timely manner. In parallel, it aims to reduce the grading workload for teaching staff, without compromising the quality or integrity of assessment. Importantly, the intervention does not seek to fully automate the grading process. Instead, auto-grading is used as a pedagogical aid to support student learning and foster iterative problem-solving. Final assessment decisions remain the responsibility of the teaching staff, ensuring alignment with learning outcomes and maintaining academic standards.

This paper reports on the design, implementation, and evaluation of this intervention, addressing the following research questions:

RQ1: What changes in grading workload and feedback turnaround are observed after introducing partial auto-grading with immediate feedback, compared with the previous course iteration?

RQ2: What changes occur in resubmission rates and in assignment marks on the redesigned tasks?

RQ3: How do students and TAs perceive the usefulness and usability of the auto-grading workflow (e.g., confidence, error detection, clarity of tests, installation/setup)?

It is worth noting that this course is a mandatory third-year course for students in the Mechanical Engineering bachelor's program and an elective for those in Industrial Design Engineering. Additionally, it is a popular choice among exchange students, with approximately 20–30 enrolling each year. Students from these backgrounds often do not identify as "programmers," and, according to the initial course survey, around half express limited interest in coding. This underscores the importance of designing structured assignments that guide students step-by-step through the problem-solving process.

2 Related work

There is a quite extensive literature about auto-grading, or *Automatic Assessment Systems*. It is established among computer science teachers that "manually grading programming assignments is time consuming and tedious, especially if they are incorrect and incomplete" (Nikhila & Chakrabarti, 2022, Abstract), negatively affecting the teaching experience. Automatic assessment systems have been developed to alleviate these issues, improving both the efficiency and effectiveness of grading.

Early studies, such as (Douce, Livingstone, & Orwell, 2005; Edwards & Pérez-Quiñones, 2008), show that automatic grading systems are important because they provide quick feedback to students. This immediate feedback helps students find and fix mistakes quickly, which is important for learning. A more recent study recognize the value of auto-grading, but alerts that "students are not likely to develop testing skills since they over-rely on the feedback from the auto-grader." (Mitra, 2023, Abstract).

Recent improvements have focused on making auto-grading systems better for teaching (Aldriye, Alkhalaf, & Alkhalaf, 2019). In particular, Hegarty-Kelly and Mooney (2021,

Abstract) pointed out that "lecturers can also provide instant feedback to students while keeping track of their progress and identifying where the gaps in students' knowledge are." This ability to monitor and adapt to student performance in real-time is a major benefit of auto-grading systems.

Paiva, Leal, and Figueira (2022, Abstract) discussed how assessing a program is more complex than just checking if it works correctly. They mentioned that "assessing a program is considerably more complex than asserting its functional correctness, as the proliferation of tools and techniques in the literature over the past decades indicates. Program efficiency, behavior, and readability, among many other features, assessed either statically or dynamically, are now also relevant for automatic evaluation." This shows that relying entirely on automated systems for grading can lead to misjudgment.

In summary, research on auto-grading for programming assignments covers many areas and shows ongoing improvements in technology and teaching methods. The main advantages are the immediate feedback for students and the speedup in grading for teachers. The main disadvantages are that students tend to over-rely on the feedback from the auto-grader, and it can be difficult for teachers to select the appropriate tool given the proliferation of these systems.

3 Methodology

This study evaluates the impact of integrating auto-grading into coding assignments through a comparative analysis of two consecutive iterations of the Introduction to Artificial Intelligence course. The 2023 iteration served as the baseline, utilizing traditional manually graded assignments, while the 2024 iteration featured redesigned assignments incorporating auto-grading with immediate feedback. To assess the impact of the intervention, we report quantitative data and feedback from students and TAs. This provided insight into both measurable outcomes and user experiences.

The methodology is organized into two subsections: Section 3.1 describes the implementation of the intervention, while Section 3.2 outlines the evaluation approach.

3.1 Implementation

The course includes four assignments in total, but only the first two were redesigned to incorporate auto-grading. The remaining two assignments are managed by other instructors and were therefore not modified as part of this intervention. As a result, students experienced both the traditional and the redesigned assignment formats within the same course, providing a natural point of comparison.

Tool Selection Process To select a suitable auto-grading solution, a comparative review of existing tools was undertaken. Factors considered included compatibility with Jupyter notebooks, ease of integration with the existing teaching infrastructure, i.e., Canvas, support for partial grading and detailed feedback, and the overall user experience for both instructors and students. As a result of the review, **Otter-Grader** (Pyles & UC Berkeley Data Science Education Program, 2024) has been selected as the auto-grading tool. The main reason is that, differently from cloud-based solutions, it does not require students to register to a third-party service. Moreover, Otter-Grader aligns with the philosophy of the new assignment design, that is to guide students towards the solution of the problems with intermediate steps. However, the tool does have some drawbacks. Students are required to install additional software—albeit in the form of a standard Python package—and the system can be somewhat complex for instructors to configure and use effectively.

Design of new assignments Following the selection of the auto-grading tool, the assignments were redesigned to incorporate automated feedback using Jupyter notebooks integrated with Otter-Grader. The notebook is structured into distinct questions that can build on one another, and tests are implemented for selected questions. These tests are designed to evaluate the functionality of the students' code, similarly to *unit testing* in software engineering. Some of the tests are visible to students, providing immediate feedback during their development process. Others remain hidden and are used exclusively by the teaching staff for verification purposes[1]. Some notebooks were deliberately scaffolded with step-by-step instructions and partially completed code templates, guiding students towards the intended solution. Others were presented in a more open format, leaving students with greater freedom in how to design and implement their solutions.

The implementation effort involved the development of two new assignments, each consisting of three problems, resulting in a total of six Jupyter notebooks built around the new structure. Several problems were adapted from previous years with minor modifications, while others were newly designed. In addition, a new assignment, called *Assignment n.0*, was introduced to familiarize the students with the system. As an example, Figure 1 shows the Jupyter notebook used for Assignment 0.

The overall redesign effort was somewhat greater than the usual yearly updates to the assignments, since additional attention had to be given to designing and validating the automated tests, but the extra workload was moderate and manageable.

Instructions to students Clear instructions were given to the students on how to install Otter-Grader and how to use it. For this purpose a "Getting started" page was published on Canvas, where (among other things) it was stated:

"For assignments 1 and 2 we will experiment the use of a system to give you immediate feedback on your solutions. Note that the automatic feedback is not comprehensive and the final grading will be done by the teachers. So, if you get all tests to pass, it does not mean that you have a perfect solution. The automatic feedback is only meant to help you testing your code and to give you some guidance on how to solve the problems."

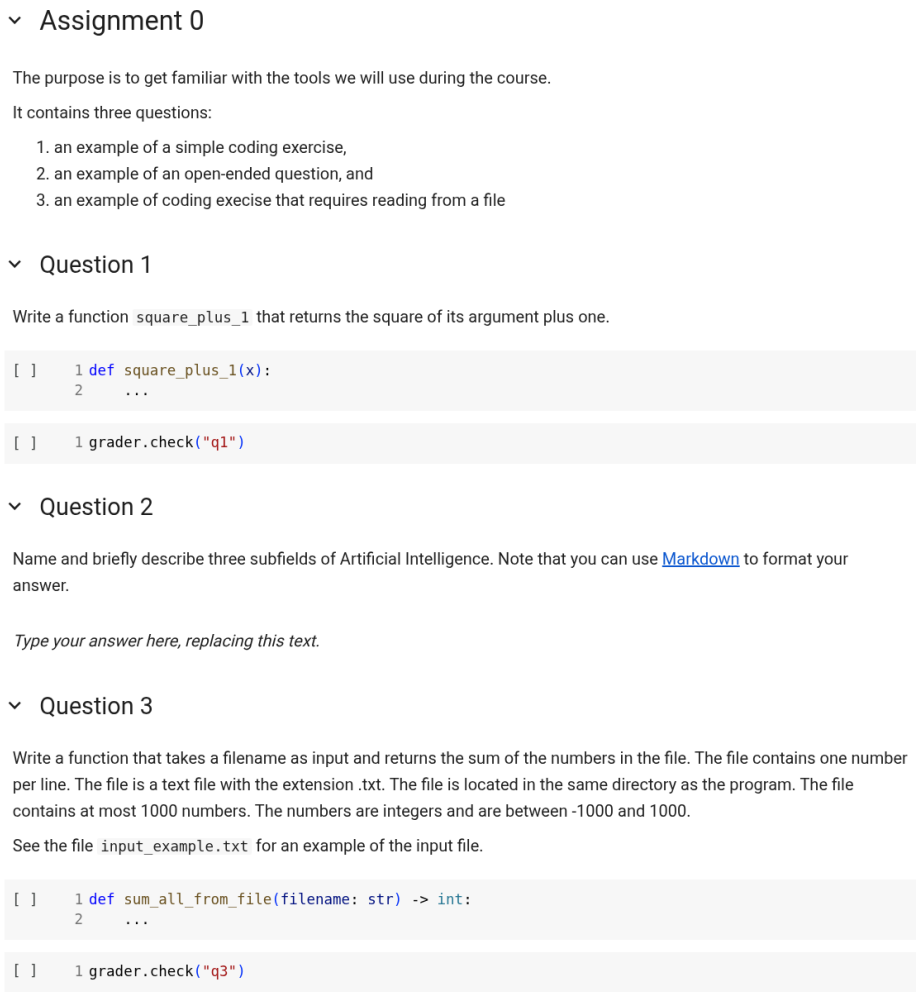
Workflow Each assignment followed a structured workflow. Students first received a Jupyter notebook containing the assignment hand-out. While working on the tasks, they could execute their code locally and obtain immediate feedback from the automated checks, allowing them to verify—before submission—that their solution passed all available tests. Once satisfied, students submitted their notebooks through the course platform (Canvas). The teaching assistants then reviewed the submissions: they re-ran the notebooks with the autograder (with additional tests), but also performed a manual inspection to evaluate aspects not fully captured by automation, such as clarity, readability, and efficiency of the code. Grades and written feedback were then provided, and students with insufficient or incorrect solutions were asked to resubmit. In this way, the auto-grader supported formative learning while final evaluation remained under instructor control.

3.2 Evaluation of the intervention

The evaluation combined course metrics with feedback from students and teaching assistants.

[1]See `https://otter-grader.readthedocs.io/en/latest/test_files/python_format.html` for more information about how to write tests in Otter-Grader.

Figure 1: The Jupyter notebook for Assignment 0. Question 1 and 3 provide automated feedback with `grader.check`; while question 2 does not.



Assignment 0

The purpose is to get familiar with the tools we will use during the course.

It contains three questions:

1. an example of a simple coding exercise,
2. an example of an open-ended question, and
3. an example of coding execise that requires reading from a file

Question 1

Write a function `square_plus_1` that returns the square of its argument plus one.

```
[ ]  1 def square_plus_1(x):
     2     ...
```

```
[ ]  1 grader.check("q1")
```

Question 2

Name and briefly describe three subfields of Artificial Intelligence. Note that you can use Markdown to format your answer.

Type your answer here, replacing this text.

Question 3

Write a function that takes a filename as input and returns the sum of the numbers in the file. The file contains one number per line. The file is a text file with the extension .txt. The file is located in the same directory as the program. The file contains at most 1000 numbers. The numbers are integers and are between -1000 and 1000.

See the file `input_example.txt` for an example of the input file.

```
[ ]  1 def sum_all_from_file(filename: str) -> int:
     2     ...
```

```
[ ]  1 grader.check("q3")
```

Course metrics. A set of indicators was collected for both iterations of the course. These included measures of workload and turnaround (average grading time per problem, time from submission to grade release), student performance (average assignment grades, percentage of required resubmissions), and overall course outcomes (number of submissions, course evaluation scores). Descriptive comparisons across the two years were used to assess changes following the introduction of auto-grading.

Student feedback. Feedback from students was gathered on three official occasions: (i) verbally during the mid-course meeting with student representatives, held after the deadline of the first two assignments; (ii) in writing through the end-of-course evaluation survey; (iii) verbally during the course board meeting with student representatives, held several months after the conclusion of the course. Students were informed that automated feedback had been introduced for the first time in their iteration of the course. Students were also informed that anonymized comments from (i) and (iii) might be used for research and reporting purposes. Participation levels varied across the three occasions: in (i), five out of six student representatives attended; in (ii), the survey response rate was 10.94% (21 out of 192 students); and in (iii), only two student representatives were present.

Teaching assistant feedback. The two teaching assistants involved in the course were also invited to reflect on their experiences with the intervention. They were explicitly informed that their reflections could be included in this report and agreed to share their feedback for that purpose. Their reflections were provided by email correspondence.

4 Results

The introduction of auto-grading yielded measurable improvements in grading efficiency, feedback timeliness, and overall user satisfaction for both students and teaching assistants. Quantitative metrics and feedback from students and TAs are reported in what follows.

4.1 Course metrics

Table 1 summarizes the course metrics before and after the introduction of auto-grading. The clearest effect was a reduction in grading time for teaching assistants: in 2023, grading a single programming problem required on average about 5 minutes, whereas in 2024 this dropped to around 3 minutes. This represents roughly a 40% improvement in grading efficiency, freeing up time for other teaching activities such as student support.

Table 1: Key metrics before (2023) and after (2024) the introduction of auto-grading

Metrics	**2023**	**2024**
Number of enrolled students	168	193
Number of submissions - Assignment 1	157	180
Number of submissions - Assignment 2	148	173
Number of resubmissions required*	15%	5%
Average grade (pts. %)	76.6%	75.2%
Average grading time per problem (minutes)	$\sim$5	$\sim$3
Time from submission deadline to grade release (days)	14	9
Course evaluation - Assessment	3.9/5	4.0/5
Course evaluation - Overall impression	3.1/5	3.6/5

*Percentage of students required to resubmit at least one assignment due to insufficient or incorrect initial submission.

Another notable change was the decrease in resubmissions. In 2023, 15% of students had to resubmit at least one assignment due to errors or incomplete work; in 2024 this proportion fell to 5%. This indicates that immediate feedback from the auto-grading system helped students detect and correct mistakes earlier, leading to higher-quality submissions on the first attempt.

Average assignment grades remained stable (76.6% in 2023 vs. 75.2% in 2024), with no statistically significant difference. This suggests that the redesigned assignments did not make the tasks easier, but rather allowed students to reach similar outcomes more efficiently.

Turnaround time between submission and release of grades also improved. Students in 2023 typically waited about 14 days for results, compared with 9 days in 2024.

Finally, the overall course evaluations improved following the intervention. Student ratings for the assessment component rose slightly (from 3.9/5 in 2023 to 4.0/5 in 2024), while the overall impression of the course increased more substantially (from 3.1/5 to 3.6/5). These scores suggest that the redesign not only improved efficiency but also contributed positively to students' perception of the course experience.

Taken together, these results point to clear operational benefits for teaching staff and a more positive learning experience for students, providing the context for the feedback reported in the following subsections.

4.2 Feedback from students

In what follows, only comments specifically related to the use of auto-grading are reported. All such comments are included, grouped thematically into three main areas: perceived learning benefits, feedback clarity, and technical aspects.

Perceived learning benefits. Several students highlighted the value of immediate automated feedback for supporting their learning. They described how the system increased their confidence and helped them identify mistakes earlier:

> *"I really like the automated feedback with Otter-Grader! It guides me towards the solution and gives me confidence that I am on the right track." (i)*

> *"The automated feedback improves my learning journey because it is immediate and the results of the automated tests helps me understanding where I made mistakes." (i)*

The inclusion of Assignment 0 was also appreciated as a low-stakes way to familiarize themselves with the tool:

> *"Nice to have Assignment n.0 to test the system." (i)*

Other students valued the ability to run tests directly and to monitor their progress:

> *"I appreciated the otter-grader function on which you could test your program directly." (ii)*

> *"I feel like the first half of the course was pretty good, with the Jupyter Notebook submissions that had these self-graders that gave a nice indication if you were on the right track or not." (ii)*

One student even emphasized that Otter-Grader should remain in the course design:

> *"What should be kept for the next round of this course? Otter grader." (ii)* and *(iii)*

Feedback clarity. Although the overall experience was positive, some students noted that feedback could be more transparent. In particular, the use of hidden tests was occasionally perceived as limiting:

> *"Good to have the expected results in the assignments. Sometimes they could be more explicit, instead of being hidden in the tests of Otter-Grader." (i)*

Technical aspects. A few students mentioned practical difficulties related to the installation of the system, pointing to the need for continued technical support:

> *"It was not my case, but I know that someone struggled to install Otter-Grader and get it working." (i)*

4.3 Feedback from teaching assistants

Two teaching assistants contributed reflections. TA-1 had experience with both the 2022/23 (manual grading) and 2023/24 (partial auto-grading) iterations of the course, while TA-2 was only involved in the latter. Their comments focused on three main areas: workload and efficiency, learning curve, and quality of feedback.

Workload and efficiency. Both TAs emphasized that Otter-Grader reduced the burden of repetitive manual corrections. TA-1 described the previous year as *"extremely challenging, requiring extensive time and effort for assignment corrections,"* whereas the new setup *"significantly eased the workload despite minor issues."* Similarly, TA-2 noted that the tool helped avoid tedious manual work and reduced the risk of errors in grading.

Learning curve. TA-1 reported that Otter-Grader initially had a steep learning curve, but once mastered, it simplified the correction process: *"Although initially Otter-Grader had a steep learning curve, it makes the corrections much easier than just manual corrections (even when it fails!)."*

Quality of feedback. The system enabled more systematic detection of errors through batch evaluation and checkpoints, while also freeing up time for richer feedback. As TA-2 put it, *"It also frees up time to give more in-depth constructive feedback to students."* At the same time, TA-1 stressed the continued need to manually open notebooks to ensure students were not bypassing the automated checks.

In summary, the teaching assistants agreed that the integration of Otter-Grader made grading more efficient and consistent, while still requiring complementary manual review.

5 Discussion and Conclusions

The intervention was designed to address the research questions by examining changes in efficiency (RQ1), submission quality and performance (RQ2), and user experience (RQ3). The findings show that partial auto-grading with immediate feedback offered both pedagogical and operational benefits in the context of a large undergraduate AI course.

Impact on learning and teaching practice

The results for RQ1–RQ3 suggest that the intervention supported student learning while also improving the grading process. Students reported that immediate feedback increased their confidence, helped them identify mistakes earlier, and enabled higher-quality submissions on the first attempt, as reflected in the lower resubmission rate. At the same time, average grades remained stable, indicating that the assignments were not made easier, but that students were able to reach comparable outcomes more efficiently. Design choices such as including a preliminary *Assignment 0* further supported onboarding and reduced technical barriers.

From the teaching perspective, grading became faster and more consistent, allowing assistants to dedicate more time to higher-value activities such as feedback and support. Reflections from TAs confirmed that Otter-Grader eased workload and provided useful tools (e.g., batch evaluation and checkpoints), though manual review was still necessary to ensure reliability. Importantly, the introduction of auto-grading also required some additional redesign effort: compared with the usual yearly updates of assignments, instructors

had to plan and validate automated tests in addition to revising the tasks themselves. This added workload was moderate, however, and considered manageable within the normal course preparation cycle, especially given the resulting gains in grading efficiency and feedback quality. Together, these results indicate that partial auto-grading can enhance both the pedagogical experience for students and the operational sustainability of large-enrollment courses.

Limitations and Challenges

Despite these positive outcomes, some limitations were noted. A few students experienced difficulties installing Otter-Grader on their local machines, highlighting the variability of student computing environments. Although these challenges are typical of software-based interventions, they underscore the need for robust technical support and clear documentation. In response, the 2025 course iteration introduced a cloud-based alternative (Google Colab with pre-installed tools) to mitigate installation issues—an important step toward greater accessibility.

Another concern, also highlighted in the literature, is the risk that students may become overly reliant on automated feedback, potentially hindering the development of essential skills such as independently testing and debugging their code. This limitation can be addressed through careful assignment design—for example, by withholding automated feedback for certain parts of the code. Such an approach encourages students to engage more deeply with the problem-solving process and to practice verifying their solutions without relying solely on external validation.

Another limitation relates to the scope of automated evaluation. While Otter-Grader provides valuable support for functional correctness, it does not fully address more nuanced aspects of programming such as code efficiency, readability, or testing strategy. To avoid misinterpretations, students were explicitly informed that the automated feedback was not equivalent to a final grade but rather a guide to support the problem-solving process.

Conclusion and Future Work

This intervention illustrates that a partial implementation of auto-grading can significantly enhance the effectiveness and efficiency of coding assignments in higher education. The approach supports formative learning, reduces grading effort, and maintains pedagogical integrity when paired with clear guidance and complementary manual review.

Future improvements could include expanding the range of exercises that use automated feedback, refining feedback messages to promote deeper reflection, and further simplifying the technical setup for students.

For educators considering similar implementations, this case highlights the importance of balancing automation with human oversight, designing user-friendly onboarding processes, and clearly communicating the role of automated tools within the broader assessment strategy.

References

Aldriye, H., Alkhalaf, A., & Alkhalaf, M. (2019). Automated grading systems for programming assignments: A literature review. *International Journal of Advanced Computer Science and Applications*, *10*(3). doi: 10.14569/IJACSA.2019.0100328

Douce, C., Livingstone, D., & Orwell, J. (2005, sep). Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, *5*(3), 4-es. doi: 10.1145/1163405.1163409

Edwards, S. H., & Pérez-Quiñones, M. A. (2008). Web-cat: automatically grading programming assignments. In *Annual conference on innovation and technology in computer science education.* doi: 10.1145/1597849.1384371

Hegarty-Kelly, E., & Mooney, D. A. (2021). Analysis of an automatic grading system within first year computer science programming modules. In *Proceedings of the 5th conference on computing education practice* (p. 17-20). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3437914.3437973

Mitra, J. (2023). Studying the impact of auto-graders giving immediate feedback in programming assignments. In *Proceedings of the 54th acm technical symposium on computer science education v. 1* (p. 388-394). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3545945.3569726

Nikhila, K. N., & Chakrabarti, S. K. (2022). Letgrade: An automated grading system for programming assignments. In M. M. Rodrigo, N. Matsuda, A. I. Cristea, & V. Dimitrova (Eds.), *Artificial intelligence in education. posters and late breaking results, workshops and tutorials, industry and innovation tracks, practitioners' and doctoral consortium* (pp. 383–386). Cham: Springer International Publishing.

Paiva, J. C., Leal, J. P., & Figueira, A. (2022, jun). Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ.*, *22*(3). doi: 10.1145/3513140

Pyles, C., & UC Berkeley Data Science Education Program. (2024, March). *Otter-Grader: A Python and R autograding solution.* doi: 10.5281/zenodo.5259955