Συνέδρια της Ελληνικής Επιστημονικής Ένωσης Τεχνολογιών Πληροφορίας & Επικοινωνιών στην Εκπαίδευση

Τόμ. 1 (2025)

14ο Συνέδριο ΕΤΠΕ «ΤΠΕ στην Εκπαίδευση»

Feedback to Students and Instructors of a Programming Course Through a Large Language Model

Aggelos Diamantopoulos, Christos Sintoris, Stavros Demetriadis, Nikolaos Avouris

doi: 10.12681/cetpe.9388

Βιβλιογραφική αναφορά:

Diamantopoulos, A., Sintoris, C., Demetriadis, S., & Avouris, N. (2026). Feedback to Students and Instructors of a Programming Course Through a Large Language Model. *Συνέδρια της Ελληνικής Επιστημονικής Ένωσης Τεχνολογιών Πληροφορίας & Επικοινωνιών στην Εκπαίδευση*, *1*, 1159–1168. https://doi.org/10.12681/cetpe.9388

Feedback to Students and Instructors of a Programming Course Through a Large Language Model

Aggelos Diamantopoulos[1], Christos Sintoris[2], Stavros Demetriadis[3], Nikolaos Avouris[2]
aggelosdiama@gmail.com, sintoris@upatras.gr, sdemetri@csd.auth.gr, avouris@upatras.gr
[1]Department of Computer Engineering and Informatics, University of Patras
[2]Department of Electrical and Computer Engineering, University of Patras
[3]Computer Science Department, Aristotle University of Thessaloniki

Abstract

This study investigates how Large Language Models (LLMs) can be used to deliver formative feedback to first-year students in an introductory programming course and to provide instructors with insights into overall class performance. It aims to evaluate the quality of feedback generated by these models and compare their performance with human evaluators on programming assignments. In addition, it explores the user experience of interacting with the AI tutor. Findings underscore the potential of advanced AI to enhance personalized learning and formative assessment in educational settings.

Keywords: Artificial Intelligence, formative assessment, Higher Education, Introduction to Programming, Large Language Models, personalized learning

Introduction

In higher education, teaching large classes with limited human resources can inhibit the learning process. This condition, especially in first year courses, results in reducing the timeliness and quality of feedback to students. To face this problem, integrating technologies such as AI Tutors based on Large Language Models represents a promising approach. We need to investigate if these technologies can generate personalized and timely feedback to individual students. Wang et al. (2024), in a survey of use of Large Language Models in education stress the transformational potential as well as the ethical, technical and pedagogical challenges that arise. According to the survey, the feedback needs to be constructive, context-relevant and free of errors and unsubstantiated claims.

This paper examines the potential of an AI Tutor in delivering feedback both to individual students and at the same time, to the instructor on overall class performance. This study is in the context of an Introductory to Programming class. Specifically, we present a proof-of-concept AI Tutor prototype that employs the GPT-4 Large Language Model to assess student lab work, tested in the case of two Python programming assignments, in which the prototype is used for offering individualized feedback on performance, highlighting key issues to the students. In addition, the AI Tutor aggregates students' feedback to generate a concise, timely report for the instructor. We describe the primary functionality and architecture of this AI tutor and report on a study designed to address two research questions:

1. RQ1: How does the AI tutor's assessment of student assignments compare with human evaluators?
2. RQ2: How informative and accurate is the feedback provided to students and instructors?

To contextualize our work, we first review current research on the use of AI tools for student feedback, then detail the design and typical usage of the AI Tutor prototype developed and we present the results of our evaluation study.

Background literature

Large Language Models and other generative AI tools are reshaping Computer Science education, as they have in software development (Li et al. 2024). Prather et al. (2023) highlight possibilities in learning, among them AI-assisted feedback. The key question is how to teach programming when AI can provide ready solutions, potentially undermining the problem-solving skills students need. Begum et al. (2025) emphasize the need for a new pedagogy to teach abstraction in computing and STEM. LLMs can support this by engaging students in reflective dialogue through feedback, rather than supplying only a 'correct' solution (Begum et al. 2025).

Examples of LLM use in teaching and learning have been reported. Kazemitabaar et al. (2024) describe CodeAid, designed to address academic integrity concerns by avoiding direct code solutions and instead offering scaffolded guidance. Sarsa et al. (2022) explored generating programming exercises and explanations to automate educational tasks, while Albdrani and Shargabi (2023) showed ChatGPT can improve engagement and outcomes in data science education through personalized feedback. Golchin et al. (2025) found LLMs, guided by instructor rubrics, produced grades more aligned with experts than peer grading in MOOCs, inspiring our use of rubric-based prompting. In automated assessment and feedback, similar studies include Askarbekuly et al. (2024), Stamer et al. (2024), and Yeung et al. (2025). Still, concerns remain about effectiveness compared to human feedback (Er et al., 2025).

AI tutors offer promising tools, but implementation must balance benefits and challenges. Kasneci et al. (2023) note that while LLMs provide immediate, personalized feedback, educators must supervise AI use and mitigate risks like bias or overreliance. Messer et al. (2024) reviewed grading tools, finding most assess code correctness with limited feedback (e.g., "passed/failed"), offering little guidance. Savelka et al. (2023) showed GPT models cannot fully pass higher-level assessments but perform well on simpler tasks, benefiting from auto-grader feedback. Phung et al. (2023) demonstrated precise feedback on syntax errors, though with limited scope. Savelka et al. (2023b) further showed that GPT-4, compared to earlier models, achieved passing scores in typical programming course assessments, highlighting both its growing capabilities and the need for educators to rethink assessment design and AI integration. Our work extends this by proposing an AI Tutor that not only grades but also fosters deeper learning through feedback.

In this section we present the design of the AI Tutor prototype (Figure 1).

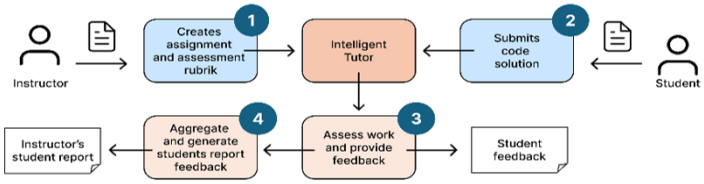

Figure 1. Intelligent Tutor: the typical use cases for students and instructor

The prototype (Figure 1) is used typically by two types of users. The instructor who provides the Tutor with the assignment and the assessment rubric (case 1), and the student, who submits their work for assessment in the form of a Python program (case 2). The Tutor then generates and delivers an assessment report as feedback to the student (case 3), and upon request, it delivers an aggregate of the student reports, as instructor feedback (case 4). By

implementing the following design, we differentiate our system from the commercial used LLM models, as it generates content only according to the assessment rubric that the instructor provided at the beginning, thus acting as tutor to the students.

The technology used for building the Intelligent Tutor is shown in Figure 2. It is a web application that has on the server side a database for storing submitted assignments and reports, while connecting to the OpenAI API, using GPT4 Large Language Model, for providing feedback to both students and instructors. As already discussed, this LLM has been used effectively for supporting Python programming, however its use in teaching and learning is still under investigation.

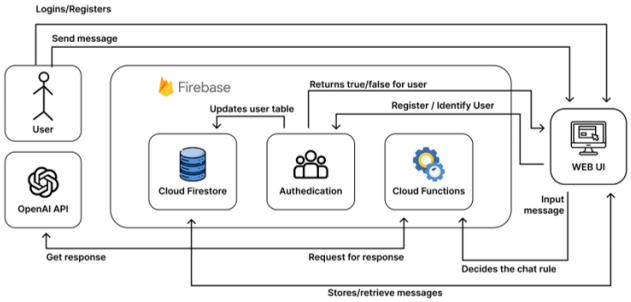

Figure 2. Intelligent Tutor Architecture

The user interface of the AI Tutor can be seen in Figure 3. In the screenshots one can see the student, submitting work (2, left) and receiving the feedback (3, right). Similar design is the user interface for the instructor, who can define an assignment and assessment instructions, and later receive comprehensive feedback on the class performance.

Figure 3. The student user interface: [2] submitting work, [3] receiving feedback

The most interesting part of this architecture is the interface with the Large Language Model API. The tutor receives the input from the instructor or the student and wraps it up to create a prompt to the LLM. A chat rule has been created for each use case as mentioned below. The chat rule is a series of prompts with instructions and directions for the system that follows the format of "act as a…" for the user persona that the system will get, series of tasks to follow and finally the format of the output. The three use cases are described below.

Use case (1): professorRule

The first API call is "professorRule" and is invoked on behalf of an instructor. As a system

directive, it contains the following instruction to the API: "*You are an expert Greek university professor with a specialty in computer language programming.*" This is the introductory part, while the rest of the prompt is the assignment description and the assessment rubric that are typed by the instructor. Therefore, the system assigns the LLM a role to produce the response accordingly. It is worth noting that the instructions were provided in English to avoid any misunderstanding by the LLM, since the API's default language is English. However, because this application is currently designed for use in Greek universities, we have specified the Greek nationality of the professor so that the message can be produced in Greek. An example of an assignment is the following (Assignment #1 of our study):
*Write a Python program where the user is asked to input an integer, followed by a space, and then either the character **b** or **x**. If the last character is **b**, the program computes the binary representation of the given number; if it is **x**, it computes the hexadecimal representation. As for the evaluation criteria, if the solution works satisfactorily with defensive programming, the grade ranges from 80% to 100%; if it lacks defensive programming, from 50% to 70%; and if it does not run at all or not produce the expected results, the score should be between 0 and 30%.*

Use case (2): studentRule

The second API call is the "studentRule" and runs when the student submits an assignment. The system instruction here defines its role as: "*You are an expert university professor specializing in computer language programming. Provide feedback to the university student on their programming language exercises.*" As before, there is an additional user-role message that provides further instructions. In this case, it is particularly important, because a student might not clearly specify how they want their feedback. So, the students just need to submit their code, which will be added to the instruction. The directive is: "*According to the following messages, get the code exercise, the evaluation criteria, and finally my solution to the code and give me feedback for improvements and mistakes and grade me from 1 to 10 according to the criteria.*" In this way, it is ensured that the student will receive informative feedback but also a grade for the submitted work.

Use case (3): professorReportRule

The final API call is the "professorReportRule," and it is triggered when all the students that have joined the session of an exercise, have submitted their code solutions. The directive is: "*As an experienced Greek university professor specializing in computer language programming, you have been provided with feedback and grades from students who completed the programming exercise. Your task is to analyze this data comprehensively and generate a report assessing how the entire group performed.*" This prompt is followed by loading all the feedback messages that each student received by the system for the specific exercise.

It then follows with the final prompt with the role of the "user", as though the user had already sent it: "*According to the previous messages, can you sum up the feedback that the students got for their code solutions. Your sum up should include what mistakes were made, theory and techniques students didn't understand and everything else that is helpful to know about their solutions. Tell me if something should be repeated in the class for better understanding. At the end tell me the mean value of the group of students without yapping. The answer should be in Greek.*" With this prompt we remind the system of what it should do with the new data that it received, because after numerous messages the GPT model may forget the initial goal. Also, we use the phrase "…without yapping," it's a slang phrase that means to avoid saying any extra words and to fill the message with cosmetic words, as the GPT model is used on doing, because it needs to sum up a big amount of information and needs to stay on point. Finally, we specify again that

the answer should be in Greek since it's targeted for Universities of Greece as it was mentioned before.

At the moment this rule can't be triggered just by the professor requesting the system, via a message, to get the report as it needs to load all the student's feedback that are not part of the professor's message history until that point.

Experimental use of the Intelligent Tutor

In order to test the developed prototype, we simulated assessment of twenty (20) randomly selected students' authentic lab work that were part of the Introductory to Python lab of a first year course at the University of Patras. These assignments were submitted by the students as part of final lab exercise.

Students were required to complete their work in the lab within one hour. They had no internet access; they were asked to submit their assignments through the institution's Learning Management System (LMS). Later, the teaching assistants manually graded the submissions, providing a grade and a brief comment as feedback. Typically, it takes around a week for students to receive this feedback via the OpenEclass LMS. Gikandi, et al. (2011) in their review of formative assessment in online contexts emphasize key factors of successful feedback: *timeliness*, *constructiveness*, *learner autonomy*, and *iterative improvement*. These principles inform how AI-driven feedback systems should be designed—for instance, ensuring that the feedback is not only quick but also contextually rich and reflective. The AI Tutor met these requirements. The feedback on average was produced and delivered to the student in less than 2 min, and it contained constructive comments for improvement. Typical feedback to a student with poor performance was as follows:

The effort you made to implement the project's requirements is a good starting point, but there are many serious issues and omissions that need fixing and improvement. Let's examine them in detail:

Conversion Functions (to_bin and to_hex)

1. *The to_bin and to_hex functions share the same error. Both take input from the list lista by removing its last two elements, which is incorrect and does not cover the necessary conversion operations for every input from the user.*
2. *Even more important is the fact that there is no check on the value to be converted. The functions simply proceed with the conversion for any arbitrary value, without verifying its validity or range.*

Function 'menu'

1. *The execution flow is disordered because the logic for handling each user input is not correct. For example, using lista[1] == 3 is not valid logic and does not serve its intended purpose.*
2. *There is no proper handling of user input for choosing between 'b' or 'h', and essentially the function just calls to_bin or to_hex without verifying that choice correctly.*
3. *There is no appropriate management of user input or the execution of the conversion functions.*

Proposed Improvements

1. *Each user input must be correctly implemented, taking the appropriate values as parameters.*
2. *The menu function should be developed with proper input handling, ensuring that user input is checked, the 'b' or 'h' character is correctly read, and then the respective conversion function is called. This functionality requires good organization and implementation both in handling user input and in executing the conversion functions. The code's level here would have been below 3 out of 10 due to significant functional shortcomings, hence the need for re-examination.*

It is evident from this example that the feedback is not in the form of a "correct" solution and a grade, as it was the case before implementing the AI Tutor, but a point-by-point comment on the submitted assignment. It should be mentioned that the Tutor grades this

work in the range 0 to 3, as the rubric provided by the instructor suggested. Several observations can be made about the AI Agent's responses to student solutions. In some well-developed submissions, the Agent provided comments suggesting further improvements to the code—additions that were not explicitly requested. In effect, it offered ideas for continued refinement and improvement.

Regarding its scoring behavior, the Agent provided different types of feedback based on the instructions it was given. For example, in the first exercise, it was instructed to use three scoring ranges (≤ 3, 5-7, 8-10). As a result, it sometimes indicated a score range rather than assigning a specific value. By contrast, in the second exercise, the Agent was told exactly how many points each question was worth, so it assigned individual scores for each question and then summed them up. This suggests that breaking the solution into separately scored components leads to more accurate scoring.

Finally, there was one instance where the Agent initially did not give a passing grade and then provided enough suggestions and comments to help the student achieve just a passing score, rather than the highest possible mark.

After the students' assignments were graded, the comprehensive feedback was delivered to the instructor. This is shown in Figure 4 with its translation to follow:

Summary of feedback on students' solutions:

1. ***Errors in the Code**: Common mistakes were observed regarding the execution of functions and input handling. Many students forgot to verify the correct format of inputs from the user and did not handle them properly. Additionally, several students did not grasp how to return information from functions or correctly manage loops.*
2. ***Theory and Techniques**: It appears that understanding of error handling (exception handling) and the use of functions such as split() and strip() requires further practice. Some students also struggled with using control structures for determining the program's behavior based on different inputs.*
3. ***Improving Understanding**: It would be helpful to review lessons focusing on both structured and object-oriented programming, as well as error handling in Python. Another lesson dedicated to operators and string processing methods would address many of the identified issues.*

***Average Grade for the Student Group**: Based on the recorded grades in each comment, the group's average grade is **7/10**.*

This aggregate feedback was produced in less than 5 min after submission of the last student assignment. It is evident that the instructor feedback contains a summary of class performance with advice on focusing instruction on specific issues of the subject matter.

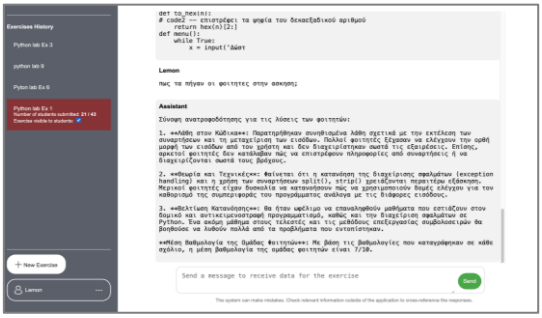

Figure 4. Instructor feedback

Following this first experimental use, a second experiment followed, when we asked the AI Tutor to assess the assignments of a group of students of an Introductory to Programming course of the University of Patras, that were part of their homework. The assignment was more complex and required submission of longer code with multiple parts. Below is an excerpt from the instructor's assignement description:

"*Submit a Python program that presents the results of local government elections in a Greek municipality. The program is divided into three parts. The first part gathers votes cast for different municipal coalitions, including invalid and blank ballots. The second part calculates percentages for each candidate based on the data from the first part. The third part announces the final results according to the votes. Use defensive programming. The grading rubric allocates 35%, 35%, and 30% of the total grade to each respective part.*"

In this case, the submitted assignments were more extended, and the students had much longer period to work, at home, where they could use aids like AI assistants. So, as will be discussed in the next section, their grades were much higher. Yet we wished to examine how the AI Tutor grades compared to the human grader in this context, as discussed in the next section.

Intelligent tutor grading

In this section, we present the grading scores of the AI Tutor for the two experiments and compare them to the human tutors, who had already graded the assignments beforehand. The overall results are shown in Figure 5. On the left are the scatter plots of human vs AI scores and on the right the plot of differences against the mean of the two scores, that helps identify systematic biases and whether the differences vary with the score magnitude.

In Assignment #1, both the human graders and the AI Tutor assigned the same average score (6.4), but the AI Tutor's grades had a smaller standard deviation. In Assignment #2, the mean scores differed slightly (9.18 versus 9.36), while their standard deviations were almost identical. This is likely because the scores for Assignment #2 were generally higher and more tightly clustered. The correlation between the human graders and AI Tutor in Assignment #1 was very strong, Pearson's $r = 0.916$, while in Assignment #2 was very strong too, Pearson's $r = 0.798$, but weaker than #1.

In the paired *t*-test we performed for the first scenario, the mean scores for both cases of students grading were, as discussed earlier, equal. The resulting *p*-value is 1.000, which is above the standard 0.05 threshold, so there is no statistically significant difference in the means. For the second exercise, the correlation coefficient (r) is 0.7979, indicating a strong positive correlation between the two raters' scores. The *p*-value is 0.00002499, which is extremely low, suggesting that such a correlation is highly unlikely to occur by chance. The 95% confidence interval for r is [0.5493, 0.9167], indicating that the true correlation likely lies within this range.

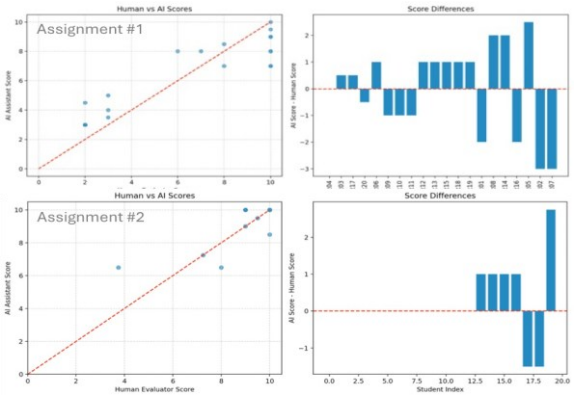

Figure 5. Scatter Plots and Bland-Altman Plots of the two experiments

The descriptive statistics of the grades for the two experiments were as shown in Figure 6.

Assignment #1			Assignment #2		
Statistic	Human	AI Tturor	Statistic	Human	AI Tutor
Mean	6.4	6.4	Mean	9.18	9.36
Median	8	7.5	Median	10	10
Std. Dev.	3.56	2.49	Std. Dev.	1.49	1.20
Range	2–10	3–10	Range	3.75–10	6.5–10

Figure 6. Descriptive statistics of the grades of the two assignments

Finally, the paired t-test for the second exercise also the p-value is 0.363, exceeding the 0.05 significance level, which means there is no statistically significant difference between the two means.

In general, the results suggest that the AI Tutor is a reliable tool for assessing student achievement. However, we have observed that the AI Tutor is less extreme than the Human graders, e.g., as it underestimates top performers and overestimates low performers. There are some extreme outliers in both examples, as seen in Figure 6 (b), with maximum difference of -3. It seems that the AI Tutor avoided extremes (lowered high scores, raised low scores) in assignment #1 while in the second case overscored low performers (e.g., +2.75 for Human 3.75) but was stricter with borderline cases (Human 8 → AI 6.5). It's worth mentioning that during the tests that we had to run the same solutions multiple times, the grades were diverging on ±1. This proves even without the paired t-test that the results are not random. Finally, we can deduct from this discussion that the AI Tutor was strongly aligned with the Human graders, but perhaps in future experiments we should calibrate the AI Tutor for extreme scores (e.g., avoid penalizing high performers), this may be achieved with more precise instructions on grading.

Conclusions

This study set out to explore the potential of the AI Tutor, powered by GPT-4, in providing formative feedback for an introductory programming course. Two separate experiments were conducted on Python assignments, comparing the AI Tutor's performance to that of human evaluators. The results highlight the feasibility of incorporating a Large Language Model-

based tutor into the grading and feedback loop for programming assignments. Next, we attempt to answer the research questions.

RQ1: The AI Tutor's grading was highly correlated with human graders in both experimental settings, demonstrating strong alignment and no statistically significant difference in the mean scores. In the first assignment, Pearson's r was 0.916, while in the second it was 0.798. Despite this strong relationship, some discrepancies appeared at the extreme ends of the grading scale, with the AI Tutor tending to slightly raise very low scores or lowering very high ones. This suggests that while the AI Tutor approximates human grading robustly, further calibration, particularly for edge cases— could enhance its alignment with human evaluators' judgments.

RQ2: The feedback generated by the AI Tutor was both detailed and timely. Students received constructive comments that went beyond merely indicating correctness, offering concrete suggestions for improvement and highlighting gaps in their code. Additionally, feedback appeared consistent with the given assessment rubric, especially when individual rubric components were clearly specified in the Tutor's prompts. Furthermore, on a focus group that was conducted with the goal to test the user experience of the platform, four (4) students participated, and they commented that the feedback of the system was very clear and detailed enough. The 4 students had previously participated in the exercises that the system was tested on, and they stated that they wish they had this kind of feedback at the time of their lab activity.

For instructors, the Tutor aggregated individual student feedback into a concise class performance summary. Within minutes of final submissions, instructors were able to see patterns in errors, common misconceptions, and average performance. This immediate feedback can enable more targeted interventions, allowing instructors to address prevalent issues in follow-up sessions rather than waiting days or weeks for manual reviews.

The study results suggest that institutions with large number of students, or large online courses, could adopt LLM-driven assessment tools to alleviate instructors' workload and ensure timely feedback for students.

On the other hand, future work could include refining the grading prompts or rubrics to reduce the tendency of the model to cluster scores around the mean, ensuring high-performing students and struggling students receive more nuanced grades.

A future direction is that, beyond simple rubric-based assessment, subsequent prototypes could integrate diagnostic questions, reflective prompts, or adaptive hints that guide students through progressive challenges—potentially fostering deeper learning and self-regulation skills. Finally, closer integration between the AI Tutor and LMSs could automate administrative tasks like recording final grades or grouping students with similar misconceptions for targeted remedial sessions.

Overall, the findings highlight the promise of Large Language Models in supporting both personalized student feedback and efficient, high-level insights for instructors, suggesting a strong potential for broader adoption in higher education settings, bearing in mind the limitations of such approach, as observed by Er et al (2024).

References

Albdrani, R. N., & Al-Shargabi, A. A. (2023). Investigating the Effectiveness of ChatGPT for providing personalized learning experience: A case study. *International Journal of Advanced Computer Science & Applications*, *14*(11), 1208-1213.

Askarbekuly, N., & Aničić, N. (2024). LLM examiner: automating assessment in informal self-directed e-learning using ChatGPT. *Knowledge and Information Systems*, *66*(10), 6133-6150.

Begum, M., Crossley, J., Strömbäck, F., Akrida, E., Alpizar-Chacon, I., Evans, A., Gross, G. B., Haglund P., Lonati, V., Satyavolu, C., & Thorgeirsson, S. (2025). A pedagogical framework for developing abstraction skills. In *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 258-299). ACM.

Er, E., Akçapınar, G., Bayazıt, A., Noroozi, O., & Banihashem, S. K. (2024). Assessing student perceptions and use of instructor versus AI-generated feedback. *British Journal of Educational Technology*, *00*, 1-18

Gikandi, J. W., Morrow, D., & Davis, N. E. (2011). Online formative assessment in higher education: A review of the literature. *Computers & Education*, *57*(4), 2333-2351.

Golchin, S., Garuda, N., Impey, C., & Wenger, M. (2025). Grading Massive Open Online Courses using Large Language Models. *Proceedings of the 31st International Conference on Computational Linguistics* (COLING) (pp. 3899-3912). Association for Computational Linguistics. https://aclanthology.org/2025.coling-main.263/

Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Gunnemann, S., Hullermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet O., Sailer, M., Schmidt, A., Stadler, M., …, & Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274.

Kazemitabaar, M., Ye, R., Wang, X., Henley, A. Z., Denny, P., Craig, M., & Grossman, T. (2024). CodeAid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs. *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (pp. 1-20). Association for Computing Machinery.

Li, Z. S., Arony, N. N., Awon, A. M., Damian, D., & Xu, B. (2024). *AI tool use and adoption in software development by individuals and organizations: A grounded theory study*. arXiv preprint. https://doi.org/10.48550/arXiv.2406.17325 .

Messer, M., Brown, N. C., Kölling, M., & Shi, M. (2024). Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, *24*(1), 1-43.

Phung, T., Cambronero, J., Gulwani, S., Kohn, T., Majumdar, R., Singla, A., & Soares, G. (2023). *Generating high-precision feedback for programming syntax errors using Large Language Models*. arXiv prerpint. https://arxiv.org/abs/2302.04662

Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., Craig, M., Keuning, H., Kiesler, N., Kohn, T., Luxton-Reilly, A., MacNeil, Ast., Petersen, A., Pettit, R., Reeves, B. N., & Savelka, J. (2023). The robots are here: Navigating the generative ai revolution in computing education. *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 108-159). ACM.

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic generation of programming exercises and code explanations using large language models. *Proceedings of the 2022 ACM Conference on International Computing Education Research* (vol. 1, pp. 27-43). ACM. https://doi.org/10.1145/3501385.3543957

Savelka, J., Agarwal, A., An, M., Bogart, C., & Sakr, M. (2023b). Thrilled by your progress! Large language models (GPT-4) no longer struggle to pass assessments in higher education programming courses. *Proceedings of the 2023 ACM Conference on International Computing Education Research* (vol. 1, pp. 78-92). ACM.

Savelka, J., Agarwal, A., Bogart, C., Song, Y., & Sakr, M. (2023). Can Generative Pre-trained Transformers (GPT) pass assessments in Higher Education programming courses? *Proceedings of the 28th Annual ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE) (vol. 1, pp. 117-123). ACM. https://doi.org/10.1145/3587102.3588792

Stamper, J., Xiao, R., & Hou, X. (2024). *Enhancing LLM-based feedback: Insights from intelligent tutoring systems and the learning sciences. Proceedings of the International Conference on Artificial Intelligence in Education* (pp. 32-43). Springer.

Wang, S., Xu, T., Li, H., Zhang, C., Liang, J., Tang, J., Yu, P. S., & Wen, Q. (2024). Large language models for education: A survey and outlook. arXiv prerpint. https://doi.org/10.48550/arXiv.2403.18105

Yeung, C., Yu, J., Cheung, K. C., Wong, T. W., Chan, C. M., Wong, K. C., & Fujii, K. (2025). *A zero-shot LLM framework for automatic assignment grading in higher education*. arXiv prerpint. https://doi.org/10.48550/arXiv.2501.14305