# Enhancing Student Focus and Problem-Solving with Real-Time LLM Feedback on Compiler Errors

Maciej Pankiewicz[1,2][0000-0002-6945-0523] and Ryan S. Baker[1][0000-0002-3051-3232]

[1] University of Pennsylvania, Philadelphia PA, USA
[2] Warsaw University of Life Sciences, Warsaw, Poland
mpank@upenn.edu

**Abstract.** Programming syntax is highly complex, which has made it consistently difficult to automate feedback for compiler errors in educational settings. Large language models (LLMs) show promise for addressing this issue at scale by providing personalized feedback tailored to specific code submissions, but the effectiveness of this feedback remains uncertain. This study evaluated the impact of GPT-4o at generating real-time feedback for compiler errors during a randomized controlled trial. A total of 248 CS1 students participated, submitting 22,674 pieces of code to an automated programming assessment platform. Students in the Experimental group received LLM feedback, while the Control group did not. Results showed that students who received LLM feedback rated it highly for usefulness, and submitted fewer non-compiling code attempts. These students also had significantly improved performance in terms of resolving errors in consecutive attempts compared to the Control group. Affective surveys revealed that the LLM feedback group self-reported higher focus and lower levels of "confrustion" (a combination of confusion and frustration) after encountering compiler errors. When LLM feedback was temporarily disabled, students in the Experimental group solved programming problems more quickly and demonstrated significant improvement in resolving errors across attempts. However, no significant differences were observed between groups in terms of final scores on a simulated exam. These findings suggest that LLM-generated feedback can improve students' coding experience, engagement, and problem-solving efficiency in the initial phases of computer science education, though it may not lead to better final performance.

**Keywords:** Automated Assessment, CS1, Large Language Models, LLM, GPT, Compiler Errors, Automated Feedback.

## 1 Introduction

Feedback is an important feature of self-paced learning environments [31]. Over the years, it has become a key component of various digital learning platforms, including intelligent tutors [14, 31], online courseware [8, 10], and specialized environments for programming education [9, 23, 27]. Despite wide adoption, its effectiveness varies and studies on the effects of feedback have yielded mixed results: some indicate sig-

nificant gains in learning outcomes and engagement [1, 34], while others highlight potential issues such as misuse or overuse [2].

Crafting a feedback message to help students fix an error in programming code is a complex task. There are three distinct and very broad error categories: syntax (compiler) errors (when code does not compile), runtime errors (when the code compiles but an exception occurs while the program is running), and logic errors (when the program runs but fails to produce the expected result). Generating feedback for each of these areas poses unique challenges. More detailed overview of feedback in programming education can be found in [20].

This paper focuses on automating feedback for compiler (syntax) errors in an introductory programming course (CS1), because the ability to fix them directly impacts early programming success [5, 11]. Fixing a syntax error may be challenging, because compiler messages are often unclear or overwhelming [33], making it difficult for beginners to understand the source of the problem. But generating meaningful feedback that precisely addresses the issue is complicated because of the extensive number of possible syntax errors, variations in message content across different compilers for the same language, the unique nature of individual student submissions, and the fact that interpreting such feedback may require a level of programming knowledge that novices may not yet possess.

Large language models (LLMs) offer promising avenues for tackling these challenges, because of the vast repositories of programming data used to train them [8] and capacity for contextual awareness [28].

Several recent studies have attempted to evaluate the effectiveness of large language models in generating feedback for programming assignments. A common approach in this line of research involves expert evaluation of the feedback content, often resulting in high ratings of its quality [6, 15, 17, 26]. However, while expert assessments provide a valuable indication of the potential of LLMs, several challenges remain regarding their practical utility in educational settings, particularly for novice programmers in introductory courses.

First, positive expert evaluations of LLM-generated feedback do not necessarily indicate that such feedback effectively supports learners with limited programming experience. Prior research has shown that level of expertise influences how effectively LLM-generated assistance can be used [13, 22]. In an experimental lab study with CS1 students, feedback generated by GPT-4 for compiler errors in the C programming language only occasionally outperformed standard compiler error messages. In contrast, human-written explanations surpassed both LLM-generated and conventional messages in both objective and subjective evaluations [32]. Similarly, a two-month study conducted in a CS1 course using C# reported mixed results for GPT-4-generated feedback [25].

Second, while some evidence points to the potential benefits of long-term usage of LLM-generated feedback, such conclusions have typically been drawn from more advanced courses [24]. These courses assume a certain level of prior knowledge, which likely influences how students interpret and act upon feedback. Given that understanding feedback may be closely tied to programming experience, further investigation is needed to determine whether similarly positive long-term effects can be

achieved within a CS1 context, especially among students with minimal or no prior programming background.

Third, much of the current research on LLMs in programming education has focused on their implementation as interactive assistants, where students engage in dialogue with the model [4, 18]. While promising, this approach poses significant integration challenges. Many introductory computer science courses rely on structured pedagogical designs, such as scaffolding and worked-out examples. Transitioning to an assistant-based model requires substantial redesign of the course structure and careful validation of its pedagogical effectiveness. As an alternative, integrating LLM-generated feedback into existing automated assessment platforms may offer a more manageable yet still impactful solution – though not without its own challenges.

Overall, past work suggests that a learner's programming experience may significantly affect the effectiveness of LLM feedback for compiler errors, and its long-term impact on the early stages of programming education warrants further investigation. Thus, it remains an open question whether such feedback can consistently support novices at this stage of their learning.

Therefore, in this study, we focus on two main research questions: (1) How do novice CS1 students perceive the quality of LLM-generated feedback delivered on-the-fly for compiler errors? and (2) What are the long-term effects of this feedback on student performance within a platform and learning outcomes? We specifically investigate whether the limitations previously identified in LLM feedback for compiler errors can be mitigated by using GPT-4o, a more recent and advanced model. To this end, we conducted an experimental study using an automated assessment platform, which delivered GPT-4o-generated feedback for student submissions that contained compiler errors.

## 2 Methods

Data for this study was collected during the first two months of the four-month winter semester of 2024/2025. The participants were computer science students enrolled in a mandatory first-semester course titled "Introduction to Programming" at a large European university. The course focused on programming in C#. A total of 248 students consented to participate and completed a pre-test (8 items, 1 point each) designed to objectively assess their programming knowledge at the start of the semester.

We utilized the RunCode platform [23] for automated assessment of programming assignments, sharing a total of 149 programming problems with students. Usage of the platform was not mandatory, and performance did not count towards the course grade. The following topics were covered: types and variables (33 problems), conditional statement (28), recursion (28), loops (17), and loops with arrays (43).

Assignment difficulty varied, with the easiest problems such as multiplying two integers, to much more difficult, such as extracting and alphabetically sorting words from a given text. All modules containing programming assignments were initially locked (for both groups), requiring students to first solve a sequence of several micro-tasks – simple coding tasks that focus on one particular aspect of the code and require

the user to enter (or edit) only one line of the code. They were designed to help students understand how different programming components work one-by-one. An example of such a task could be: "use the = operator and assign a value to the variable *age*". Though completion of the micro-tasks was required to advance to the programming assignments, neither was required as part of the course grade.

The platform's assessment process involved compiling and testing the submitted code. Submissions that failed to compile were marked as *non-compiling*. Feedback included compiler messages for non-compiling code submissions. For compiling submissions students received results for each unit test executed on the code (with input values and expected outcome for each unit test), and overall score (the percentage of unit tests that ended successfully: 0-100%).

While solving assignments on the platform students self-reported their affective states through a dynamic HTML element presented directly after they received submission results with the question: "Choose the option that best describes how you feel at the moment" with the following response options: *Focused*, *Anxious*, *Confused*, *Frustrated*, *Bored*, *Other* (in this order) and appropriate emoticons to visually highlight each of the available responses. This set of affective states was chosen based on their importance for learning [12]. In previous research, it has been found that unresolved confusion and frustration negatively impact learning outcomes in similar ways [16, 19], and theoretical work has argued that extensive empirical evidence demonstrates that the two affective states form a confrustion constellation, where it matters less whether a student is specifically confused or frustrated than details of the experience (such as resolution, duration, interpretation) which produce common results across both affective states [21, 29]. Hence, we decided to combine these affective states prior to our study (but still asked students to self-report using commonly understood terms). To avoid irritation that could arise from being required to fill out the affect survey too frequently, it was not presented after every submission, but randomly, with a probability of 1/10.

Students in the Experimental (GPT) condition additionally received feedback generated automatically using the OpenAI's GPT-4o model for submissions containing compiler errors. Feedback was created via an API request using the approach to prompt generation described in [25] and consisted of the following parts: 1) general instructions for an assistant, 2) text of a programming assignment, 3) student code, 4) list of detected compiler errors. Feedback examples were not added to the prompt, which resulted in more detailed responses from the utilized GPT-4o model. Feedback was displayed to students as a pop-up immediately after submission. Once the pop-up was closed, the feedback remained accessible below the code editor for further review. To evaluate the usefulness of GPT-generated feedback, students in the Experimental group were also presented with a dynamic HTML element prompting them to rate the usefulness of the feedback they received. The rating was collected using a 5-point Likert scale, ranging from "1 – Not useful at all" to "5 – Extremely useful." This survey appeared after every fifth instance of GPT feedback provided to the student. To assess programming competency, a non-graded simulated exam was organized in the middle of the semester (after 2 months), featuring two programming assignments drawn from previous exams: the first focused on loops and numerical arrays, and the

second on loops and string operations. These assignments were made available on the platform, and students had the opportunity to complete them within a fixed 60-minute timeframe from home, similar to their regular use of the platform. While multiple attempts were allowed, the platform only displayed the total score and did not provide details of the unit tests executed on the code. However, if compiler errors were detected, they were presented to students, just as they would be if the tasks were solved locally on their computers. For these assignments GPT feedback was not provided.

# 3 Results

In this section, we analyze the reception of the GPT feedback and examine its impact on affect based on student reports after the submission of code containing compiler errors. We further assess student performance on the platform, and for assignments with GPT feedback disabled for the Experimental group.

## 3.1 Pre-test scores

To verify that both groups had comparable baseline knowledge prior to the intervention, a Mann-Whitney U test was conducted to compare pre-test scores between the Experimental (N=121, Mdn=4) and Control (N=127, Mdn=4) groups. The results showed no statistically significant difference in pre-test performance between the two groups, (W=7502, p=0.748).

## 3.2 Usefulness of the GPT feedback

To evaluate the perceived usefulness of the GPT generated hints, a total of 1036 responses were collected from N=62 students within the Experimental (GPT) group throughout the duration of the study (as a reminder, use of the tool was not mandatory). The Control group did not have access to the GPT feedback; therefore, no responses were recorded from this group. Figure 1 displays the histogram of the frequency of students according to the median rating they gave to received feedback, with the x-axis representing the median of the feedback rating for a student. The majority of students (81%) rated the feedback they received as *very* or *extremely* useful – this shows a positive reception of the GPT feedback feature by students. The usefulness of feedback was rated significantly higher than the neutral midpoint of 3 (Mdn=4.5), as indicated by a one-sample Wilcoxon signed-rank test (W=1424.5, $p<0.001$).

## 3.3 Affective states

To understand the impact of the GPT feedback feature on student affect, we performed an analysis of responses in the affect survey displayed after receiving the results for the submission. The most frequently reported state in both conditions was *focused* (56%). A total of 2788 survey responses were collected (Experimental: 1288,

Control: 1500) from 117 students (Experimental: 60, Control: 57) during the period of study.

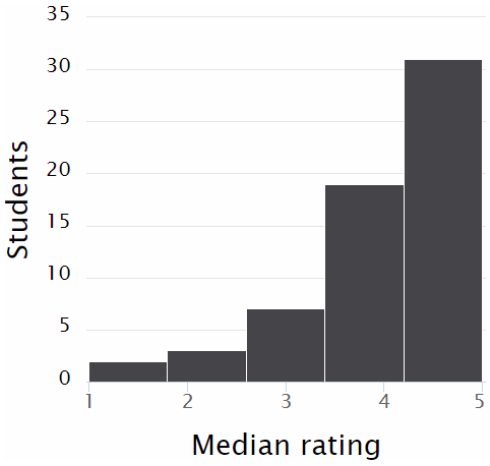



**Fig. 1.** Distribution of students according to the median rating for received GPT feedback.

Out of these responses, 1300 (Experimental: 583, Control: 717) were reported directly after receiving feedback for a non-compiling submission. In the following analysis we excluded students with less than 3 responses. To compare differences in the proportions of the affective states reported by each student after a non-compiling submission, non-parametric Mann-Whitney U tests were calculated, due violation of normality assumptions. To control the false discovery rate resulting from multiple comparisons, the Benjamini-Hochberg alpha correction was applied. Two affective states showed marginally significant differences between conditions. For *focused*, students in the Experimental group reported higher rates (Mdn=0.66, N=50) than those in the Control group (Mdn=0.4, N=46), (W=1471.5, adjusted $\alpha$=0.0125, p=0.018, for a non-parametric Mann-Whitney U test). For *confrustion*, students in the Experimental group reported lower rates (Mdn=0.13) than those in the Control group (Mdn=0.22), (W=866.5, adjusted $\alpha$=0.025, p=0.037, for a non-parametric Mann-Whitney U test). For *anxiety*, the Experimental group reported a median that was not significantly different (Mdn=0, Mean=0.04) from the Control group (Mdn=0, Mean=0.06), (W=1054, adjusted $\alpha$=0.05, p=0.364, for a non-parametric Mann-Whitney U test). The same was true for *boredom* (Exp. Mdn=0, Mean=0.05 vs. Contr. Mdn=0, Mean=0.04), (W=1188.5, adjusted $\alpha$=0.075, p=0.694, for a non-parametric Mann-Whitney U test). The means reflect that there was some small amount of anxiety and boredom despite medians being 0.

### 3.4 Student progress on the platform

During the study period, students performed 22,674 code submissions for programming assignments in modules: types and variables, conditional statement, arrays and

loops: in the Control group: 12,226 (N=78), and in the Experimental (GPT) group: 10,448 (N=71), with an average of 152.2 submissions per student (Contr.=157, Exp.=147.1). There were 9,304 (41%) non-compiling submissions (Contr.=5,369, Exp.=3,935). The number of students who submitted code at least once is lower than the total number of students in the experiment, as not all participants joined the platform.

To understand the impact of the automated GPT feedback on student progress within the platform, we analyzed the rate at which students completed programming assignments. We compared the number of successfully solved tasks within the first 200 submissions on the platform, averaging the number of tasks solved per student in each condition. Students supported by GPT feedback for compiler errors seemed to perform slightly better in terms of the number of completed assignments within the first 200 submissions (Figure 2).

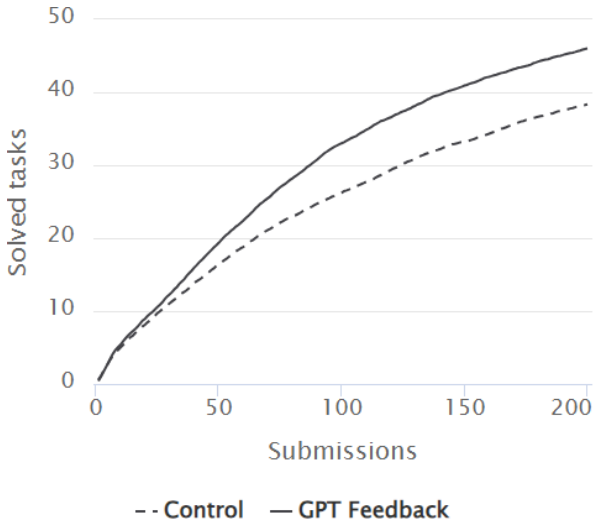



**Fig. 2.** Relationship between the number of submissions and tasks solved for GPT Feedback and Control Groups (first 200 submissions).

We also compared the total number of completed assignments between both groups. This difference was not statistically significant (GPT Mdn=50.5, Control Mdn=41, W=2900.5, p=0.426 for a non-parametric Mann-Whitney U test).

A rank-based regression analysis was conducted to assess the impact of GPT feedback on the frequency of submissions containing compiler errors, controlling for pre-test scores. The results revealed a significant reduction in the average frequency of non-compiling submissions for participants in the GPT group compared to the Control group ($\beta=-0.049$, $t(145)=-2.01$, $p=0.047$). Pre-test scores also had a significant negative association with the frequency of errors ($\beta=-0.041$, $t(145)=-7.35$, $p<0.001$), indicating that higher pre-test scores were linked to fewer errors. The model explained approximately 29.3% of the variation in the frequency of compiler errors

($R^2_{robust}$=0.293) and was statistically significant overall (F=29.47, p<0.001). These findings suggest that availability of GPT feedback contributed to a measurable reduction in submissions containing compiler errors.

To better understand the trajectory of the intervention's impact, we also compared the frequency of non-compiling submissions across the first 7 attempts on each task for both groups. Frequency was calculated as the ratio of non-compiling submissions to the total number of submissions. For each attempt, we calculated error frequency cumulatively including all submissions up to this attempt. A linear mixed-effects model was conducted to examine the effects of group (GPT vs. Control), attempt, and their interaction on the frequency of non-compiling submissions. The random intercept for participants accounted for variability between students (σ2=0.03), and residual variability was small (σ2=0.002). The results showed that the GPT and Control groups did not differ significantly in their mean frequency of non-compiling submissions at the first attempt (GPT: Mean=0.43; Control: Mean=0.39; b=0.040, SE=0.029, t(146)=1.36, p=0.175). Additionally, there was no significant main effect of attempt (b=0.001, SE=0.001, t(886)=1.45, p=0.148), suggesting that the frequency of non-compiling submissions did not consistently change across attempts. However, a significant interaction between group and attempt (b=−0.011, SE=0.001, t(886)=−7.44, p<0.001) indicated that the GPT group improved over time, with frequency of non-compiling submissions decreasing significantly faster than in the Control group (Figure 3). These findings highlight the effectiveness of the intervention in reducing non-compiling submissions over consecutive attempts.

### 3.5 Performance after GPT feedback disabled

To understand the potential impact of GPT feedback feature on student performance even after it is no longer available, we analyzed submissions made on a non-graded simulated exam, featuring two programming assignments drawn from previous exams: the first (T1) focused on loops and numerical arrays, and the second (T2) on loops and string operations. For these tasks, GPT feedback was disabled for the Experimental group.

N=82 students attempted the first task (T1) with 39 students correctly solving the assignment (Control: 21, Experimental: 18) and 74% average score. N=55 students (Control: 27, Experimental: 28) attempted the second task (T2) and 21 students correctly solved the assignment (Control: 9, Experimental: 12) with 55% average score.

We calculated the total score as the sum of the highest scores obtained for each task and conducted a rank-based regression analysis to examine the effect of group assignment on total scores, while controlling for pre-test scores. The pre-test scores had a positive and significant effect on final scores (β=0.100, t(80)=4.667) p<0.001), indicating that higher pre-test scores are associated with higher final scores. However, assignment to the GPT group (β=−0.082, t(80)=−0.799, p=0.427) did not show a statistically significant effect on final scores.

To evaluate differences in terms of non-compiling submissions, we conducted a rank-based regression analysis to examine the effect of group assignment on the frequency of non-compiling submissions, while controlling for pre-test scores. The anal-

ysis revealed that pre-test scores had a significant negative effect on the frequency of non-compiling submissions (β=-0.095, t(80)=-8.099, p<0.001), indicating that higher pre-test scores were associated with fewer non-compiling submissions. In contrast, assignment to the GPT group did not have a statistically significant effect (β=0.043, t(80)=0.772, p=0.442).

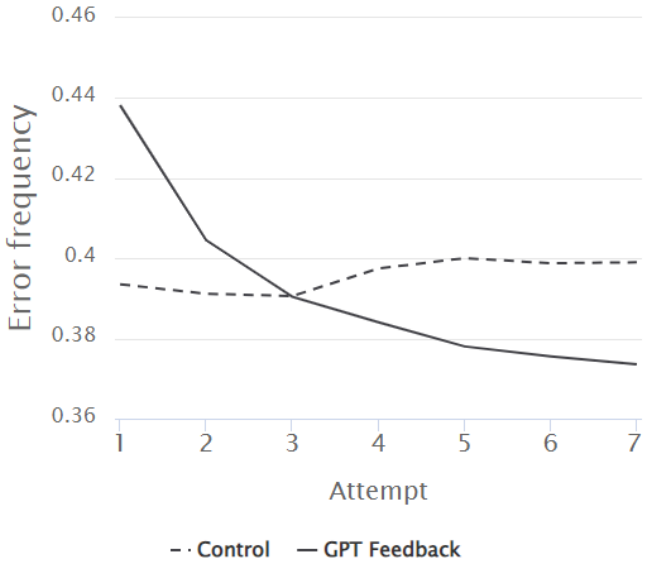



**Fig. 3.** Error frequency across consecutive attempts for GPT Feedback and Control Groups (assignments with GPT feedback enabled).

A linear mixed-effects model was conducted to examine the effects of group (GPT vs. Control), attempt, and their interaction on the frequency of non-compiling submissions. The random intercept for participants accounted for variability between students (σ2=0.14), and residual variability was small (σ2=0.01). The results indicated that there was no significant difference in frequency of non-compiling submissions between the GPT group (Mean=0.57) and the Control group (Mean=0.48; b=0.084, SE=0.083, t(83)=1.02, p=0.312). However, there was a significant main effect of attempt, with frequency of non-compiling submissions decreasing over time (b=−0.010, SE=0.003, t(508)=−3.67, p<0.001). Importantly, a significant interaction was found between group and attempt (b=−0.013, SE=0.004, t(508)=−3.12, p=0.002), indicating that the GPT group improved at a faster rate than the Control group (Figure 4).

These findings suggest that even after the GPT feedback was not available, students in the Experimental condition had a reduced frequency of non-compiling submissions over time compared to the Control condition.

### 3.6 GPT feedback disabled – time to solve

We further compared both groups in terms of time performance and conducted a rank-based regression analysis to examine the time required to solve each assignment (T1 and T2), controlling for pre-test scores.

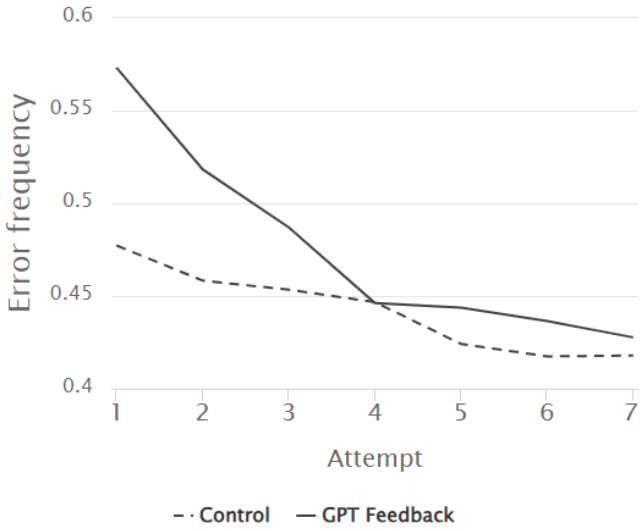



**Fig. 4.** Error frequency across consecutive attempts for GPT Feedback and Control Groups (GPT feedback disabled).

For the first task (T1), the analysis revealed that condition significantly predicted the time needed to complete the assignment, with participants in the GPT condition showing a significant reduction in completion time compared to those in the Control condition ($\beta=-567.81$, $t(36)=-2.832$, $p=0.007$). Pre-test scores, however, did not have a significant effect on ranked assignment completion time, ($\beta=-38.448$, $t(36)=-0.888$, $p=0.38$).

To complement the rank-based regression results, we calculated the means of completion times: 1645s (27m) for the Experimental (GPT) group and 1992s (33m) for the Control group. While rank-based regression is robust to outliers, its coefficients reflect relative differences in ranks rather than absolute values, limiting interpretability on the original scale. Means provide a clearer sense of the practical differences between groups.

For the second task (T2), we applied the same rank-based regression method. The results indicated a marginal effect of group membership, with participants in the GPT condition showing a potential (marginally significant) reduction in ranked completion time compared to the Control group ($\beta=-642.94$, $t(18)=-2.037$, $p=0.057$). Pre-test scores again did not significantly predict ranked completion time ($\beta=-92.93$, $t(18)=-0.985$, $p=0.337$). The mean completion time for the GPT condition was 1489s (25m), while the Control group had a mean completion time of 1854s. (31m). These findings suggest that, after controlling for pre-test scores, condition was a significant

factor influencing assignment completion time in terms of rank order, with participants in the GPT condition generally completing the assignment faster than those in the Control group. On average, in the GPT group it took 20m 22s to perform the first submission for task T1, and 21m 43s in the Control condition. For task T2, GPT condition needed 22m 43s, and Control group 26m 21s to perform the first submissions. For both tasks, consecutive submissions happened earlier in the GPT condition than in the Control group. Results are presented on Figure 5.

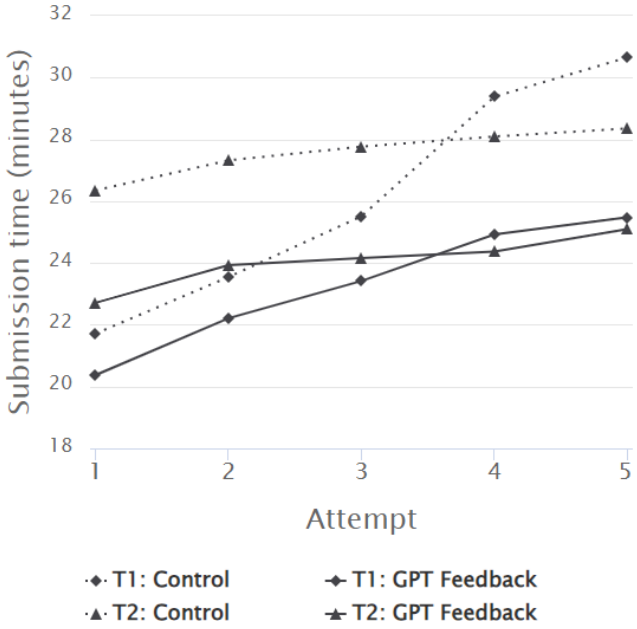



**Fig. 5.** Submission time across attempts for tasks T1 and T2 in GPT Feedback and Control Groups.

These findings suggest that, after controlling for pre-test scores, the earlier presence of GPT feedback was a significant factor influencing later assignment completion time (without GPT feedback), with participants in the GPT group generally completing the assignment faster than those in the Control group. This finding is similar to [24], where a significant improvement in terms of time needed to solve was observed for assignments on which GPT feedback was disabled.

## 4 Discussion

In this study, GPT-4o was used to automate feedback generation for non-compiling submissions on the platform for automated assessment of programming assignments. According to survey responses, a significant majority of students (81%) rated the usefulness of feedback positively and only 8% did not find it beneficial. The student

ratings were significantly higher than the neutral midpoint and better than in past studies involving GPT feedback for programming assignments [24, 25].

Compared to previous studies, our method differed in several ways: 1) the use of the GPT-4o model, improved compared to the older GPT-4 and GPT-3.5 models, 2) a more extended intervention, involving a wide range of assignments of various difficulty, 3) rather than turning off GPT feedback at the halfway point and evaluating performance after that point (as in [25]), we tested the effectiveness of the intervention on a separate set of tasks, at the same time for all students.

We found that students in the GPT condition significantly less frequently submitted non-compiling code when GPT feedback was available. Additionally, the frequency of non-compiling submissions decreased significantly faster in GPT group. For assignments with GPT disabled for the Experimental group, this effect persisted, and the GPT condition again resulted in significantly faster decrease in the frequency of non-compiling submissions across consecutive attempts, compared to the Control group. Additionally, we observed that students in the GPT condition needed significantly less time to complete assignments when GPT was disabled. However, these shorter-term benefits did not result in higher final scores on a simulated exam for the GPT condition, and neither condition was at ceiling, suggesting that the GPT condition was not successful at producing better learning that transferred to an external task.

According to student self-reports, the availability of GPT feedback led to significant differences in terms of higher levels of focus and lower levels of *confrustion*. This was a positive outcome of the intervention – according to the literature, increased focus has been often found to be positively associated with learning [7, 30], while *confrustion* has a more complex and often negative relationship to learning [3].

Previous studies on GPT feedback for programming assignments in CS1 showed mixed results. It has been reported that GPT-4 feedback for the C language outperformed standard compiler messages only in 1 out of 6 tasks, measured by student's time-to-fix each problem, and handwritten explanations outperformed the LLM messages [32]. According to [25], GPT-4 messages for C# language also were not equally good in addressing all compiler errors and did not effectively help students resolve errors requiring deeper understanding of data types.

In this study, we utilized the newest GPT-4o model, to investigate whether it led to better results than its predecessor. We offered GPT feedback for a larger set of CS1 assignments of more varied difficulty. We attempted to provide a thorough evaluation of the observed impact from many different angles: feedback reception (perceived usefulness), effectiveness (ability to improve non-compiling submissions in consecutive attempts), and learning transfer and outcomes (if the effect of the intervention persists to later tasks where the feedback is no longer given). Our study demonstrates the potential of the GPT-4o model to significantly enhance the learning experience in introductory programming courses by producing a persisting reduction of non-compiling submissions, fostering focus, and decreasing confusion/frustration, while suggesting that the ultimate goal of better learning outcomes has still not been reached.

This study has several limitations that will necessitate further investigation. It was conducted over a period of only two months, covering half of the four-month semester. Future research should investigate the long-term effects of GPT feedback on student performance and learning outcomes across semesters or academic years. Another limitation is the sample size and single setting, which may impact the robustness and generalizability of the findings. Future studies should replicate this intervention with larger and more diverse participant groups to enhance external validity. Additionally, this study focused solely on C#, while introductory computer science courses often involve other programming languages, such as Java and C++. Expanding future research to include a wider range of programming languages and more demanding courses, such as data structures and algorithms, would provide a more comprehensive understanding of the efficacy and applicability of GPT feedback across different CS contexts.

Overall, this study suggests that using an LLM to automate feedback generation can offer benefits for both performance and emotional engagement in the initial phases of programming education, potentially reducing barriers and making CS education more accessible. However, there remains considerable room for improvement.

## References


1. Aleven, V. et al.: Example-tracing tutors: Intelligent tutor development for non-programmers. Int. J. Artif. Intell. Educ. 26, 224–269 (2016).
2. Aleven, V. et al.: Help helps, but only so much: Research on help seeking with intelligent tutoring systems. Int. J. Artif. Intell. Educ. 26, 205–223 (2016).
3. Baker, R.S. et al.: The Confrustion Constellation: A New Way of Looking at Confusion and Frustration, https://osf.io/4rgtk. https://doi.org/10.35542/osf.io/4rgtk.
4. Bassner, P. et al.: Iris: An AI-Driven Virtual Tutor for Computer Science Education. In: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. pp. 394–400 Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3649217.3653543.
5. Becker, B.A. et al.: Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education. pp. 177–210 Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3344429.3372508.
6. Chapagain, J. et al.: A Study of LLM Generated Line-by-Line Explanations in the Context of Conversational Program Comprehension Tutoring Systems. In: Ferreira Mello, R. et al. (eds.) Technology Enhanced Learning for Inclusive and Equitable Quality Education. pp. 64–74 Springer Nature Switzerland, Cham (2024).
7. Fancsali, S.: Causal discovery with models: behavior, affect, and learning in cognitive tutor algebra. In: Educational Data Mining 2014. (2014).
8. Finnie-Ansley, J. et al.: The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In: Proceedings of the 24th Australasian Computing Education Conference. pp. 10–19 ACM, New York, NY,

USA (2022). https://doi.org/10.1145/3511861.3511863.
9. Gabbay, H., Cohen, A.: Exploring the Connections Between the Use of an Automated Feedback System and Learning Behavior in a MOOC for Programming. In: Hilliger, I. et al. (eds.) Educating for a New Future: Making Sense of Technology-Enhanced Learning Adoption. pp. 116–130 Springer International Publishing, Cham (2022).
10. Goldin, I. et al.: Hints: you can't have just one. In: Educational Data Mining 2013. (2013).
11. Jadud, M.C., Dorn, B.: Aggregate Compilation Behavior. In: Proceedings of the eleventh annual International Conference on International Computing Education Research. pp. 131–139 ACM, New York, NY, USA (2015). https://doi.org/10.1145/2787622.2787718.
12. Karumbaiah, S. et al.: How does Students' Affect in Virtual Learning Relate to Their Outcomes? A Systematic Review Challenging the Positive-Negative Dichotomy. In: Proceedings of the International Learning Analytics and Knowledge Conference. pp. 24–33 (2022).
13. Kiesler, N. et al.: Novice Learners of Programming and Generative AI - Prior Knowledge Matters. In: Proceedings of the 24th Koli Calling International Conference on Computing Education Research. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3699538.3699580.
14. Koedinger, K.R., Corbett, A.: Cognitive Tutors. Cambridge Handb. Learn. Sci. 61–78 (2005). https://doi.org/10.1017/CBO9780511816833.006.
15. Koutcheme, C. et al.: Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. pp. 52–58 Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3649217.3653612.
16. Lee, D.M.C. et al.: Exploring the relationship between novice programmer confusion and achievement. In: Affective Computing and Intelligent Interaction: 4th International Conference, ACII 2011, Memphis, TN, USA. pp. 175–184 (2011).
17. Leinonen, J. et al.: Using Large Language Models to Enhance Programming Error Messages. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. pp. 563–569 Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3545945.3569770.
18. Liu, R. et al.: Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. pp. 750–756 Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3626252.3630938.
19. Liu, Z. et al.: Sequences of Frustration and Confusion, and Learning. In: Proceedings of the 6th International Conference on Educational Data Mining. pp. 114–120 (2013).
20. Lohr, D. et al.: You're (Not) My Type- Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks? J. Comput. Assist. Learn. 41, 1, e13107 (2025). https://doi.org/https://doi.org/10.1111/jcal.13107.
21. Mogessie, M. et al.: Confrustion and gaming while learning with erroneous examples in a decimals game. In: Artificial Intelligence in Education: 21st International Conference, AIED 2020. pp. 208–213 (2020).

22. Moradi Dakhel, A. et al.: GitHub Copilot AI pair programmer: Asset or Liability? J. Syst. Softw. 203, 111734 (2023). https://doi.org/https://doi.org/10.1016/j.jss.2023.111734.
23. Pankiewicz, M. et al.: Using Intelligent Tutoring on the First Steps of Learning to Program: Affective and Learning Outcomes. In: Artificial Intelligence in Education: 24th International Conference, AIED 2023. pp. 593 – 598 (2023). https://doi.org/10.1007/978-3-031-36336-8_92.
24. Pankiewicz, M., Baker, R.S.: Large Language Models (GPT) for automating feedback on programming assignments. In: 31st International Conference on Computers in Education Conference Proceedings, Volume I . pp. 68–77 (2023).
25. Pankiewicz, M., Baker, R.S.: Navigating Compiler Errors with AI Assistance - A Study of GPT Hints in an Introductory Programming Course. In: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. pp. 94–100 Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3649217.3653608.
26. Phung, T. et al.: Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. In: Feng, M. et al. (eds.) Proceedings of the 16th International Conference on Educational Data Mining. pp. 370–377 International Educational Data Mining Society, Bengaluru, India (2023). https://doi.org/10.5281/zenodo.8115653.
27. Price, T.W. et al.: iSnap: towards intelligent tutoring in novice programming environments. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on computer science education. pp. 483–488 (2017).
28. Radford, A. et al.: Language models are unsupervised multitask learners. OpenAI blog. 1, 8, 9 (2019).
29. Richey, J.E. et al.: Gaming and Confrustion Explains Learning Advantages for a Math Digital Learning Game. In: Proceedings of the International Conference on Artificial Intelligence and Education. (2021).
30. Rodrigo, M.M.T. et al.: Affective and behavioral predictors of novice programmer achievement. In: Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education. pp. 156–160 Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1562877.1562929.
31. Roll, I. et al.: Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. Learn. Instr. 21, 2, 267–280 (2011).
32. Santos, E.A., Becker, B.A.: Not the Silver Bullet: LLM-enhanced Programming Error Messages are Ineffective in Practice. In: Proceedings of the 2024 Conference on United Kingdom \& Ireland Computing Education Research. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3689535.3689554.
33. Traver, V.J.: On compiler error messages: what they say and what they mean. Adv. Hum.-Comp. Int. 2010, (2010). https://doi.org/10.1155/2010/602570.
34. VanLehn, K.: The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. Educ. Psychol. 46, 4, 197–221 (2011).