

A Preliminary Analysis of the Relationship between Novice Programmers' Feedback Use and Performance within the RunCode Learning Platform

Arnel OCAY ^{a, e*}, Arthur NEBRAO JR. ^b, Maciej PANKIEWICZ ^c, Ryan S. BAKER ^d, & Maria Mercedes RODRIGO ^e

^a*Urdaneta City University, Philippines*

^b*Ateneo de Davao University, Philippines*

^c*University of Pennsylvania, USA*

^d*Adelaide University, Australia*

^e*Ateneo de Manila University, Philippines*

[*arnelocay@ucu.edu.ph](mailto:arnelocay@ucu.edu.ph)

Abstract: We investigate the impact of LLM-generated feedback on first-year students' programming activities by analyzing interaction logs from the RunCode platform. We collected log data from 37 students across 22 programming exercises. Data were collected asynchronously over five weeks as part of an introductory Java programming course at a university in the Philippines. In this preliminary analysis of the data, we attempted to answer two research questions: What was the relationship between students' feedback usage and the student performance on the next problem? What was the relationship between students' use of feedback and future programming performance? Our results show that feedback usage was not a statistically significant predictor of next-task performance, next-task error count, or future mean task scores, but it was a statistically significant positive predictor of future mean error count. These findings suggest that access to feedback alone does not translate into measurable improvements in programming performance, and the positive association with future error counts likely reflects underlying learner difficulty rather than a detrimental effect of feedback.

Keywords: LLM-generated feedback, novice programmers, computing education

1. Introduction

Feedback plays a critical role in supporting learning, as it enables error diagnosis, promotes reflection, and provides scaffolding that helps learners regulate their problem-solving strategies and progressively build understanding (Jackson & Graesser, 2007; Hull & du Boulay, 2015). Adaptive immediate feedback can increase novice programmers' engagement and intention to persist in computer science, reflecting benefits primarily in affective and short-term performance outcomes (Marwan et al., 2020). However, no amount of feedback will be useful if students choose not to use it or fail to interpret it correctly. A seminal found that many students are unable to relate compiler error messages to their mistakes, partly because of limited knowledge and partly because the messages themselves were poorly constructed (Becker et al., 2016). More recent studies of automated feedback systems report that novice learners often struggle to meaningfully interpret detailed corrective feedback or automated program repairs, suggesting that highly prescriptive assistance may reduce opportunities for essential cognitive activities such as debugging, tracing, and problem-solving (Kurniawan et al., 2023).

The emergence of Large Language Models (LLMs) offers an opportunity to provide novices feedback that is more easily understood. LLMs are deep neural networks designed

to process complex linguistic structures (Carlini et al., 2021). They have attracted significant attention within the field of natural language processing (Pankiewicz & Baker, 2023). In recent years, the emergence of LLMs, most notably OpenAI's GPT series (Brown et al., 2020; Radford et al., 2019), has enabled the generation of personalized and context-aware feedback at scale. This capability has positioned LLMs as promising tools in educational contexts, where they are increasingly explored for their potential to enhance learning experiences (Finnie-Ansley et al., 2022; Dai et al., 2023; Pardos & Bhandari, 2023). As a result, personalized feedback is widely recognized as an integral component of effective instructional scaffolding, supporting learners' conceptual understanding and skill development (Jackson & Graesser, 2007; Hull & du Boulay, 2015).

We undertake this study to determine how novice programmers engage with increasingly powerful forms of automated support, such as LLM-generated feedback, employed in an actual classroom setting. This paper studies how novice programmers made use of feedback within a platform with automated assessment of programming assignments and seeks to answer the following research questions:

(RQ1) What was the relationship between students' feedback usage and the student performance on the next problem?

(RQ2) What was the relationship between students' feedback usage and future programming performance?

1.1 The Use of LLM-based Feedback in Programming Education

Introductory programming courses (CS1) are widely recognized as cognitively demanding for novice learners, who frequently struggle with syntactic accuracy, logical reasoning, and debugging processes (Robins, Rountree, & Rountree, 2003; Watson & Li, 2014; Winslow, 1996). Studies have shown that many first-year students are unable to produce correct program code (McCracken et al., 2001) and that novice programmers often struggle to understand and interpret program code in classroom settings (Lister et al., 2004). These persistent difficulties emphasize the need for instructional systems that can respond to learners during problem-solving in a programming context.

In computing education, new forms of instructional support for learners have been introduced. Concurrently, a growing body of research has emphasized the importance of personalizing feedback to enhance student learning outcomes (Deeva et al., 2021). In this context, studies have begun to explore how large language models (LLMs) can be leveraged to automate the generation of such feedback. For example, experimental evidence shows that employing an LLM (OpenAI's GPT-3.5) to generate personalized hints for students on an automated assessment platform led to higher rates of successful submissions and was positively rated by learners, suggesting that LLM-based feedback can effectively support student problem-solving on programming assignments (Pankiewicz & Baker, 2023). However, the availability of LLM hints may also introduce a risk of over-reliance, as learners initially struggled more when hints were removed, indicating that careful integration is needed to balance assistance with independent skill development (Pankiewicz & Baker, 2023).

As a result, LLMs potentially attract attention in both computing education and software development (Takerngsaksiri et al., 2023), as they offer promising capabilities for automating programming-related tasks and providing on-demand assistance to learners (White et al., 2023). However, despite the increasing use and availability of LLM-based feedback in programming education, there remains a limited empirical understanding of how learners engage behaviorally with such feedback in actual classroom settings, and whether accessing this feedback translates into immediate or sustained performance improvements. This study attempts to address these gaps by analyzing students' behavioral interaction within an LLM-assisted learning platform using actual programming course contents.

1.2 RunCode Platform

RunCode is a cloud-based programming platform designed to help students learn and practice fundamental coding concepts. The platform integrates Large Language Model (LLM)-based assistance, enabling the delivery of context-aware hints, explanations, and feedback to learners who encounter difficulties while solving programming tasks. The platform has been deployed in a university setting since 2017 and was originally developed for students enrolled in programming courses at the Warsaw University of Life Sciences (Pankiewicz, Baker & Ocumpaugh, 2023).

RunCode has been used in several contexts to study student engagement, learning behavior, and the effects of automated and AI-supported feedback in programming education. Specifically, prior work has employed RunCode to examine how gamification elements (e.g., progress indicators and achievements) influence students' task completion patterns and engagement in introductory programming courses, revealing differential effects for beginners and students with prior programming experience (Pankiewicz, 2020). More recently, RunCode has also been used as an experimental platform to investigate the integration of LLMs to generate automated, personalized hints for programming assignments, allowing researchers to analyze performance across attempts, feedback usage, time-on-task, and potential over-reliance on AI-generated feedback (Pankiewicz & Baker, 2023). RunCode feedback was shown to increase student performance and persistence (Zhou et al., 2025), increase student engagement, and lower levels of confusion and frustration. These prior explorations using RunCode were conducted in universities in developed countries. We were interested in conducting similar studies in a developing country context to see whether the effects of the platform and of LLM-supported feedback were similar. Testing a system in a culturally different context helps determine how transferable it is. Testing AI-powered systems within under-resourced educational institutions is particularly interesting, as results give insight into how equitably such resources can be shared.

1.3 RunCode Customization for the Study

The RunCode problem sets used in this study were customized to align with the local curriculum and pedagogical standards. The problem sets used were co-designed with faculty members teaching CS1 in the higher education institution, to ensure their compliance and validity to be used in the experiment. Each programming task was designed with clearly defined specifications, including a task title, a detailed task description, and starter code. RunCode utilizes function-based programming tasks and validates student submissions through predefined test cases. These test cases were defined during the creation of the programming assignments. Figure 1 illustrates an example programming task used in this study and deployed via RunCode.

In the process of solving a task, students may have needed to submit more than one possible solution. A task was therefore associated with one or more submissions. RunCode executed each submission and checked its outputs against predefined test cases. A submission was considered correct only if it successfully passed all test cases. If a submission was incorrect, RunCode generated feedback for compiler or logical errors. The feedback, however, was not automatically displayed. Rather, students were only given an indicator that the feedback was available. Students had to click the feedback (bulb) icon labeled "Hint". Only then would RunCode show the feedback (see Fig 1).

The feedback of RunCode was generated using OpenAI's API (GPT-5 model) using a predefined prompt in response to compilation errors, runtime errors, or unit test failures in the student's code. An example of such feedback was shown in Figure 1, which was derived from an error made during submission. Once displayed, the feedback can be toggled on or off by clicking the arrowhead icon located on the same line as the Hint icon. RunCode logs these toggle actions, which are interpreted as an indication that the student accessed the feedback.

The screenshot displays the RunCode platform interface for a task titled "CSCI21 Task 01 Add". The task description asks the user to complete a function that accepts two positive integers, *a* and *b*, and returns the sum. The code editor shows a Java class `Operations` with a method `Add` that is partially implemented. A submit button is located below the code editor. The interface also shows the user's score (0%), an errors list, and a hints section. A feedback panel is visible at the bottom, detailing the error found on line 6: a missing semicolon at the end of the `sum = a + b` statement. The feedback includes a hint and an additional note about the importance of punctuation in Java.

Task Title and Description

Coding area for solution

Submit Button

Score and Error list area

Hint icon that can be toggled on and off (show/hide)

Feedback

Figure 1. Programming task illustrating a student error and the corresponding LLM-generated feedback provided in the RunCode platform.

2. Methods

2.1 Participants

Participants were first-year, first-semester computer science students at a private higher education institution in the Philippines. Inclusion criteria required participants to be at least 18 years old at the time of data collection and to be officially enrolled in Introduction to Programming (CS1), the first major programming course in their curriculum. The course introduced foundational programming concepts and was a requirement subject for students at this year level. Following the university's curriculum, all programming problems were for Java.

A total of 126 students were enrolled in CS1. Of these, 33 declined to participate, 93 initially provided informed consent to participate in the study; however, one participant later withdrew. Of the remaining 92 participants, only 37 were included in the final analysis, as they both accessed feedback and completed the exercises in the prescribed order. These two factors were essential for the analysis. Participation was voluntary, and platform usage did not count toward the final grade.

2.2 Data Collection Procedure

All programming activities were conducted using RunCode asynchronously. The platform automatically logged all student interactions during all the programming tasks. RunCode produced a submission-level dataset in which each row corresponded to a single code submission with a unique ID. Each submission row included student account identifiers, corresponding task IDs, timestamps, and compilation and testing results. Along with the submission-level dataset, RunCode also produced datasets for the hints clicked dataset, hint contents dataset, affect dataset, difficulty level dataset, and other task-related and engagement-related datasets.

All students were assigned RunCode user accounts. To preserve the students' anonymity, only the teachers knew the student-user account mapping. The research team only had access to the user accounts and therefore could not trace RunCode activity back to specific students. The logs of all students who did not consent to be part of the study were removed during data preprocessing. The final dataset enabled analysis at multiple levels, such as the submission, problem, and student levels, including attempts and LLM-supported behaviors.

The data collection process took place over 5 weeks from October 2025 to November 2025. A total of 22 RunCode programming tasks were made available. The scope of these exercises was data types, operations, conversions, decision structures, and iterative constructs. Conduct of the study had formal ethics approval from the University's research ethics office.

3. Analysis

To determine the relationship between feedback use and task difficulty, as indicated by student error, we conducted a forward-looking analysis. For each task n , students were divided into feedback users and non-feedback users based on whether they accessed feedback for at least one submission for that task. For each group, we then computed average performance on all subsequent tasks ($n + 1$ to the final task). Performance on each task was operationalized using two complementary measures derived from the RunCode submission logs:

- *Error occurrence* was a task-level indicator with a value of 0 if there were no task submissions with errors and 1 if there was at least one submission with an error. For each student, the mean future error rate was computed as the proportion of subsequent tasks ($n + 1$ to the last task attempted) in which at least one error occurred. Higher values indicate poorer performance.
- *Task score* or percent of tests passed was defined as the final correctness score recorded by the system for each task, expressed as the proportion of tests passed (range: 0 to 1). This measure allowed for partial credit when solutions passed some but not all tests. For each student, the mean future task score was computed as the average of the final task score across all subsequent tasks ($n + 1$ through the final task).

Note that the task score was not the inverse of error occurrence, nor was it defined as the reciprocal of the number of attempts. While error occurrence captured the presence of non-compiling submissions, task score reflected solution correctness, and the two measures capture distinct aspects of performance. To avoid confounding effects due to task order, the analysis was restricted to 37 students who, as mentioned in section 2.1, (a) accessed feedback at least once and (b) completed the tasks in the same prescribed sequence.

Furthermore, all variables used in the analysis were operationalized as follows.

- *Feedback usage* was defined as a binary indicator at the task level, where a value of 1 indicates that a student accessed feedback (i.e., toggled the hint interface at least

once) for any submission within that task, and 0 otherwise. This measure reflected feedback access rather than confirmed engagement with or application of the feedback.

- *Feedback opportunities*, on the other hand, refer to instances in which feedback was made available to the student, typically following incorrect submissions, where the system indicated that feedback could be accessed.
- *Task score* was defined as the final correctness score recorded by the learning platform for a given task, expressed as the proportion of test cases passed (e.g., range: 0 to 1). This score corresponded to the student's last submission for that task.
- *Error occurrence* was defined as a task-level binary indicator, with a value of 1 if at least one submission for the task resulted in a compilation or logical error, and 0 if all submissions were error-free.
- *Error count* was defined as the total number of compiler errors and warnings generated across all submissions for a given task.

The final dataset included logs from 37 students who completed two sets of programming exercises for two instructional groups of topics. The first set comprised 12 exercises, while the second set included 10 exercises, resulting in a total of 22 exercises administered across the cohort. In summary, we collected 1188 total submissions and 144 total feedback requests. Moreover, during the experiment, the task completion rate was 96.6%, the feedback request rate was 11.8%, approximately 1 in 9 actions was a feedback request, and the average number of attempts per task was 2.17. These indicators were used to analyze student feedback access and performance within the learning platform.

3.1 RQ1: What was the relationship between students' feedback usage and the students' performance on the next problem?

To examine whether using feedback on a given task was associated with immediate changes in subsequent performance, we estimated mixed-effects lagged regression models predicting performance on task $n + 1$ as a function of feedback use on task n . Because students completed multiple programming tasks, the mixed-effects models accounted for repeated observations within students. The lagged design ensured that feedback usage on task n temporally preceded the performance outcome on task $n + 1$, thereby reducing concerns about simultaneity bias.

Feedback usage was operationalized as a binary indicator at the task level:

$$\text{FeedbackUsed}_{i,n} = \begin{cases} 1 & \text{if student } i \text{ accessed at least one feedback item on task } n \\ 0 & \text{otherwise} \end{cases}$$

Two complementary measures of performance were examined:

- **Next-task score**, defined as the proportion of passed test cases on task $n + 1$:
 $\text{Score}_{i,n+1} \in [0, 1]$
- **Next-task error count**, defined as the total number of compiler errors and warnings on task $n + 1$. Because this variable was positively skewed, a log transformation was applied:

$$\log(\text{ErrorCount}_{i,n+1} + 1)$$

For each outcome, we estimated a mixed-effects regression model of the form:

$$Y_{i,n+1} = \beta_0 + \beta_1 \text{FeedbackUsed}_{i,n} + \beta_2 Y_{i,n} + \beta_3 \log(\text{ErrorCount}_{i,n} + 1) + \gamma_{\text{task}} + u_i + \epsilon_{i,n}$$

where:

- $Y_{i,n+1}$ is either the next-task score or the next-task error count

- γ_{task} represents task fixed effects
- u_i is a random intercept for student i , accounting for repeated observations
- $\epsilon_{i,n}$ is the residual error term

Feedback usage on task n was not a statistically significant predictor of next-task performance, after controlling for current performance ($\beta = -0.020$, $p = 0.166$). Similarly, feedback usage did not significantly predict next-task error count ($\beta = 0.069$, $p = 0.566$). The models explained a moderate proportion of variance in outcomes (next-task score: Marginal $R^2 = 0.252$, Conditional $R^2 = 0.277$; next-task error count: Marginal $R^2 = 0.354$, Conditional $R^2 = 0.385$), indicating that while the overall models capture meaningful variance, feedback usage itself was not a significant contributor. These results suggest that simply accessing feedback is not associated with measurable improvements in subsequent task performance or error reduction. The small and non-significant effect sizes further indicate limited practical impact, implying that feedback access alone may be insufficient to produce short-term gains, and that how students engage with and apply feedback likely plays a more critical role.

3.2 RQ2: What was the relationship between students' feedback usage and future programming performance?

To assess whether feedback usage was associated with longer-term performance trends, we extended the lagged approach by examining students' performance across all subsequent tasks following a given task n .

For each student-task pair, future performance was computed using all tasks completed after task n :

- Future mean score:

$$\text{FutureMeanScore}_{i,n} = \frac{1}{K} \sum_{k=n+1}^N \text{Score}_{i,k}$$
- Future mean error count:

$$\text{FutureMeanError}_{i,n} = \frac{1}{K} \sum_{k=n+1}^N \text{ErrorCount}_{i,k}$$

Where:

- N is the final task completed by student i
- $K = N - n$ is the number of future tasks completed

As with RQ1, the future mean error count was log-transformed prior to modeling.

The same mixed-effects structure was used, replacing the next-task outcome with the corresponding future mean outcome:

$$Y_{i,n}^{\text{future}} = \beta_0 + \beta_1 \text{FeedbackUsed}_{i,n} + \beta_2 Y_{i,n} + \beta_3 \log(\text{ErrorCount}_{i,n} + 1) + \gamma_{\text{task}} + u_i + \epsilon_{i,n}$$

Feedback usage on task n was not significantly associated with future mean task scores after controlling for current performance ($\beta = -0.007$, $p = 0.569$). However, feedback usage was a statistically significant positive predictor of future mean error count ($\beta = 0.124$, $p = 0.037$). The models explained a modest proportion of variance at the fixed-effects level but substantially more when accounting for random effects (future mean score: Marginal $R^2 = 0.084$, Conditional $R^2 = 0.642$; future mean error count: Marginal $R^2 = 0.116$, Conditional $R^2 = 0.789$). These results suggested that while accessing feedback was not associated with improved future performance in terms of scores, it was associated with higher error counts across subsequent tasks. The small effect size for scores and the positive association with errors may have indicated that feedback access alone did not translate into improved learning outcomes and may have reflected underlying task difficulty, learner struggle, or selective use of feedback by students encountering challenges.

4. Conclusions

This study analyzed observable behavioral patterns within an AI-assisted programming learning platform and their association with performance outcomes. Interpreting these findings required careful consideration of how key variables were operationalized. In particular, feedback usage reflected access to feedback rather than direct evidence of students' engagement with or understanding of that feedback. Therefore, the present analyses should be interpreted as measuring observable feedback access rather than whether students read, understood, or successfully applied the feedback. As a result, the log-based design captured observable associations between behavior and performance but provided only a partial view of the underlying learning processes, which may include unobserved cognitive and metacognitive factors.

The results indicated that feedback usage was not a statistically significant predictor of next-task performance, next-task error count, or future mean task scores, but it was a statistically significant positive predictor of future mean error count. Taken together, these findings suggested that simply accessing feedback does not lead to immediate or sustained improvements in programming performance as measured by scores. The positive association with future error counts likely reflected underlying learner difficulty; students who are struggling may have been more inclined to access feedback, rather than a negative effect of feedback itself. Overall, the small or non-significant effect sizes pointed to a limited direct impact of feedback access on performance outcomes. These findings should be interpreted in light of the study context, where tasks may have been relatively accessible, and students had prior exposure to foundational programming concepts, potentially reducing the observable impact of feedback. More importantly, the results highlighted that access alone is insufficient; the effectiveness of feedback likely depends on how students interpret, process, and apply it during problem-solving.

We continue to investigate interactions between students and feedback, and the relationship between outcomes, because it addresses broader questions about the design and implementation of learning environments for programming. In this instance, the deployment of RunCode within the Philippine university required some customization. The sequence of topics, as well as the programming conventions enforced by the Philippine teachers, prevented the use of existing RunCode exercises and necessitated the crafting of new exercises whose solutions were consistent with the implemented curriculum. Otherwise, the technical deployment and use of RunCode were problem-free. This could be attributed in part to the fact that the Philippine university and its faculty and students were relatively well-resourced. The same may not be true if we had implemented RunCode under different circumstances.

Crafting instructional interventions such as LLM-generated feedback, hints, and learning scaffolds requires substantial time and effort from educators and researchers. Although the present findings indicate that feedback access alone was not associated with immediate improvements in programming performance, they should not be interpreted as evidence that such interventions lack value. Rather, the results suggest that the effectiveness of LLM-supported feedback depends not merely on its availability but on how learners understand, engage with, and apply the feedback during problem-solving. These findings therefore motivate continued investment in designing such instructional features that are pedagogically meaningful, timely, and capable of promoting deeper cognitive engagement instead of simply increasing access to assistance. An important next step, therefore, is to examine students' perceptions of the usefulness of LLM-generated feedback, particularly in relation to their observed patterns of engagement and performance. Combining these perception data with behavioral logs and performance measures may provide a more complete understanding of how students engage with and benefit from AI-supported feedback.

5. Limitations and Future Work

While this study provides initial insights into students' use of LLM-generated feedback in an authentic programming environment, several opportunities remain for future research. First, the analyses were conducted on a subset of students who satisfied the study's inclusion criteria, which were necessary for the lagged analyses. Future studies involving larger and more diverse student populations would help establish the generalizability of the findings. Second, the present study did not consider students' prior programming ability, which may influence how learners engage with and benefit from LLM-generated feedback. Incorporating measures of prior ability could provide a deeper understanding of differences in feedback use across learners. Finally, future work may complement fine-grained measures of feedback access with richer interaction data, such as time spent on feedback, scrolling behavior, code revisions, and qualitative methods (e.g., interviews or think-aloud protocols), to provide a more comprehensive understanding of how students interpret, engage with, and apply LLM-generated feedback.

Acknowledgements

We extend our sincere gratitude to the CS1 faculty and students of Ateneo de Manila University. We thank Jessica Sugay, Gerick Jeremiah Niño Go, Roswold Jemuel Sanchez, and Reniel Gabriel Manaog for their invaluable contributions to this study.

References

- Becker, B. A., Glanville, G., Iwashima, R., McDonnell, C., Goslin, K., & Mooney, C. (2016). Effective compiler error message enhancement for novice programming students. *Computer Science Education*, 26(2-3), 148-175.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.
- Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., ... & Raffel, C. (2021). Extracting Training Data from Large Language Models. In *USENIX Security Symposium (Vol. 6)*.
- Dai, W., Lin, J., Jin, F., Li, T., Tsai, Y., Gasevic, D., & Chen, G. (2023). Can Large Language Models Provide Feedback to Students? A Case Study on ChatGPT. <https://doi.org/10.35542/osf.io/hcgzi>
- Deeva, G., Bogdanova, D., Serral, E., Snoeck, M., & De Weerd, J. (2021). A review of automated feedback systems for learners: Classification framework, challenges and opportunities. *Computers & Education*, 162, 104094.
- Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., and Prather, J. (2022). The robots are coming: Exploring the implications of openai codex on introductory programming. In *Australasian Computing Education Conference*, pp. 10–19
- Hull, A., & du Boulay, B. (2015). Motivational and metacognitive feedback in SQL-Tutor. *Computer Science Education*, 25(2), pp. 238-256.
- Jackson, G. T., & Graesser, A. C. (2007). Content matters: An investigation of feedback categories within an ITS. *Frontiers in Artificial Intelligence and Applications*, 158, 127.
- Kurniawan, O., Poskitt, C. M., Hoque, I. A., Lee, N. T. S., Jégourel, C., & Sockalingam, N. (2023). How Helpful do Novice Programmers Find the Feedback of an Automated Repair Tool? In Proc. TALE'23. IEEE, 2023 (pp. 1–6). <https://doi.org/10.1109/tale56641.2023.10398393>
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). *A multi-national study of reading and tracing skills in novice programmers*. ACM SIGCSE Bulletin, 36(4), 119–150. <https://doi.org/10.1145/1041624.1041673>
- Pankiewicz, M. (2020). Move in the Right Direction: Impacting Students' Engagement with Gamification in a Programming Course. EdMedia and Innovate Learning 2020
- Pankiewicz, M. & Baker, R. (2023). Large Language Models (GPT) for Automated Feedback on Programming Assignments. *Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education*.
- Pankiewicz, M., Baker, R., & Ocumpaugh, J. (2023). Using intelligent tutoring on the first steps of learning to program: affective and learning outcomes.

- Pardos, Z. A., & Bhandari, S. (2023). Learning gain differences between ChatGPT and human tutor generated algebra hints. arXiv preprint arXiv:2302.06871.
- Marwan, S., Gao, G., Fisk, S., Price, T., and Barnes, T. (2020). Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. *In Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. Association for Computing Machinery, New York, NY, USA, 194–203. <https://doi.org/10.1145/3372782.3406264>
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*. ACM SIGCSE Bulletin, 33(4), 125–180. <https://doi.org/10.1145/572139.572181>
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, D. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 9.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172.
- Takerngsaksiri, W., Warusavitarn, C., Yaacoub, C., Hou, M. H. K., & Tantithamthavorn, C. (2023). Students' perspective on AI code completion: benefits and challenges. <https://arxiv.org/abs/2311.00177>
- Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. *Proceedings of ITiCSE*. <https://dl.acm.org/doi/10.1145/2591708.2591749>
- Winslow, L. E. (1996). Programming pedagogy—A psychological overview. *ACM SIGCSE Bulletin*, 28(3), 17–22.
- White, J., Hays, S., Fu, Q., Spencer-Smith, J., & Schmidt, D. C. (2023). ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. <https://arxiv.org/abs/2303.07839>
- Zhou, Y., Pankiewicz, M., Paquette, L., & Baker, R. (2025). Impact of LLM Feedback on Learner Persistence in Programming. *Proceedings of the 33rd International Conference on Computers in Education (ICCE 2025)*