Enhancing Programming Error Messages in Real Time with Generative AI

Bailey Kimmel
Abilene Christian University
Abilene, Texas, United States
blk20c@acu.edu

Austin Lee Geisert
Abilene Christian University
Abilene, Texas, United States
alg22b@acu.edu

Lily Yaro
Abilene Christian University
Abilene, Texas, United States
lcy20a@acu.edu

Brendan Gipson
Abilene Christian University
Abilene, Texas, United States
bmg20b@acu.edu

Ronald Taylor Hotchkiss
Abilene Christian University
Abilene, Texas, United States
rth19b@acu.edu

Sidney Kwame Osae-Asante
Abilene Christian University
Abilene, Texas, United States
sko21a@acu.edu

Hunter Vaught
Abilene Christian University
Abilene, Texas, United States
hrv19a@acu.edu

Grant Wininger
Abilene Christian University
Abilene, Texas, United States
tgw20a@acu.edu

Chase Yamaguchi
Abilene Christian University
Abilene, Texas, United States
chy20a@acu.edu

ABSTRACT

Generative AI is changing the way that many disciplines are taught, including computer science. Researchers have shown that generative AI tools are capable of solving programming problems, writing extensive blocks of code, and explaining complex code in simple terms. Particular promise has been shown in using generative AI to enhance programming error messages. Both students and instructors have complained for decades that these messages are often cryptic and difficult to understand. Yet recent work has shown that students make fewer repeated errors when enhanced via GPT-4. We extend this work by implementing feedback from ChatGPT for all programs submitted to our automated assessment tool, Athene, providing help for compiler, run-time, and logic errors. Our results indicate that adding generative AI to an automated assessment tool does not necessarily make it better and that design of the interface matters greatly to the usability of the feedback that GPT-4 provided.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; **Empirical studies in HCI**; **User studies**; **Natural language interfaces**; **User interface programming**; • **Computing methodologies** → **Artificial intelligence**; • **Social and professional topics** → **Computing education**; **Computer science education**; **CS1**; • **Applied computing** → **Education**.

KEYWORDS

AI; Artificial Intelligence; Automatic Code Generation; Codex; Copilot; CS1; GitHub; GPT-4; ChatGPT; HCI; Introductory Programming; Large Language Models; LLM; Novice Programming; OpenAI

CHI EA '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0331-7/24/05
https://doi.org/10.1145/3613905.3647967

ACM Reference Format:
Bailey Kimmel, Austin Lee Geisert, Lily Yaro, Brendan Gipson, Ronald Taylor Hotchkiss, Sidney Kwame Osae-Asante, Hunter Vaught, Grant Wininger, and Chase Yamaguchi. 2024. Enhancing Programming Error Messages in Real Time with Generative AI. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3613905.3647967

1 INTRODUCTION

The introduction of artificial intelligence (AI) in the form of large language models (LLMs) is changing many disciplines, including computer science education [5]. Models such as GPT-3 and GPT-4 and tools such as Github Copilot have upended decades of pedagogical wisdom [1]. These tools can write, explain, and debug code in ways that simply were not possible just two years ago [19]. Researchers have been quick to measure the ability of these LLMs with regard to typical computer science programs [7, 8], code explanations [12], exams [17], and even Parsons Problems (mixed-up code problems) [24]. This has led to instructors having many concerns, such as students using autogenerated code that they don't understand [23] or using these tools to do their work for them [1, 11]. Still other concerns, such as over-reliance, biases inherent in the models, and trustworthiness, remain active points of discussion in current research [31].

Despite the alarm, generative AI has positive uses. For instructors, these models can automatically generate programming exercises [25] and code explanations [15]. They can even be used to create entirely new types of problems for students to solve, such as Prompt Problems, which ask students to write the prompt that would solve a given problem [4]. For students, generative AI can be a useful tool for learning when it's used to generate code a student already knows how to write [19], explain code or a concept that they don't understand [12], and encourage better problem definition through prompt engineering [3].

One area in which very recent strides have been made is in using generative AI to explain cryptic programming error messages

(PEMs) [13]. Students and instructors have complained for the past six decades about how difficult it can be to understand some PEMs [2]. Recent work has shown that enhancing PEMs via generative AI is very effective in actual classroom settings [27, 30]. However, these have just been for compiler error messages.

In this paper, we explore the integration of generative AI into an automated assessment tool (AAT) in a CS1 course that provides feedback on compiler errors, run-time errors, and logic errors. We added real-time feedback from ChatGPT to one of the programming assignments in CS1 during the Fall 2023 semester and report on submission statistics as well as survey responses after the assignment was completed. Our hypotheses are that students will make fewer submissions with the guidance of ChatGPT and that students will find the real-time AI feedback helpful.

We are guided by the following research questions:

RQ1: To what extent does real-time AI feedback impact student submission behavior when working on programming assignments?

RQ2: How do students perceive real-time AI feedback on their assignment submissions?

The contributions of this work are:

(1) We implement generative AI into an automated assessment tool to enhance all programming error messages at both compile time and run time, as well as logic errors, for the first time.
(2) We discuss design considerations for integrating generative AI into automated assessment tools. The implications for *how* students utilize such feedback can help both future researchers and tool creators make more usable interfaces.

2 RELATED WORK

Early work on LLMs in computing education centered on the capabilities of such tools. Already with GPT-3, which debuted in 2021, researchers found that generative AI was capable of solving common programming problems, such as the Rainfall Problem, with relative ease. Finnie-Ansley et al. found that it could solve most published variants of the problem with just the text description provided to the model. They also found that the model could answer exam questions, placing in the top quartile when compared to real student performance in a CS1 course [7]. Follow-up work revealed similar performance in a CS2 course [8]. Recent work with GPT-4 has shown that current models outperform their predecessors by getting nearly every question correct, placing this model near the top spot when compared to humans [19, 26].

Students are also using these tools to help them write code. Recent user studies on student behavior and interaction with LLMs, such as Github Copilot, have provided an interesting window into their use. Vaithilingam et al. looked at very early usage of students using LLMs to code and found that students preferred using it, despite it not decreasing their overall task completion times [29]. The fact that it didn't make novices any faster at solving programming problems could be because students struggle to understand code that has been autogenerated for them, as found by Prather et al. [23]. Kazemitabaar et al. found that students will utilize it to either explore new ways of doing a task or to attempt to accelerate their task completion time [10].

One way to help students is to use LLMs to produce code explanations. MacNeil et al. added code explanations to an ebook in a small study (n=30) and found that students rated them as understandable and helpful, though they did not engage with them as much as expected [15]. Leinonen et al. compared code explanations from students and GPT-3 in a large study (n= 1,000), rating the explanations for accuracy, understandability, and length. They found that LLM-created explanations were significantly easier to understand and more accurate than student-created explanations [12].

Another related area of support that students have consistently needed over the past six decades is in understanding the feedback they receive about their programs [2, 6, 21, 22]. Leinonen et al. used Codex (a coding model built, at the time, on GPT-3) to interpret common PEMs and found the GPT-enhanced versions often surpassed the original PEMs in understability and actionability. Although showing promising results, the model they used is now outdated and they utilized a static approach. More recent work by Wang et al. implemented real-time GPT-4 feedback on compiler error messages in a large-scale course [30]. They found that students repeated an error 23% less often and resolved the error in 36% fewer attempts, when compared to students using the original error messages. Taylor et al. implemented GPT feedback on compiler error messages into the Debugging C Compiler (DCC) in a very large course (n=2,565) [27]. They found that these messages were accurate 90% of the time and were well received by students. However, they only provided feedback for compiler error messages. Given that there may be less of a need to provide feedback on syntax errors only – because generative AI can write syntactically correct code – it is important to now move beyond PEMs and into additional feedback on student submissions.

Hellas et al. examined how GPT-3.5 would respond to novice programmer help requests [9]. They had previously collected a dataset of students who had pressed a "help" button and placed their help request into a queue for the instructor. When evaluating these requests via GPT-3.5, they noted that it often provided good feedback and almost always included code. In related work, Liffiton et al. created a tool for students (CodeHelp) to use where they could get automated help from an LLM for their coding problems. The tool allows them to paste the relevant part of their code, an error message, and a question into three separate boxes and then request help [14]. They found that students valued the tool's availability because it could be accessed when instructors or TAs were otherwise not available.

Our study extends the above work by using LLMs to providing feedback on all submissions, not just PEMs. We also extend the recent line of work on student help requests by having GPT responses to student submissions any time they submit and having that response provide feedback on their overall submission. Instead of having to ask for help or paste relevant bits of code and an error message into a tool, our implementation provides consistent help automatically.

3 METHODOLOGY

Students in the introductory programming course at Abilene Christian University in the Fall 2023 semester (n=52) were recruited to

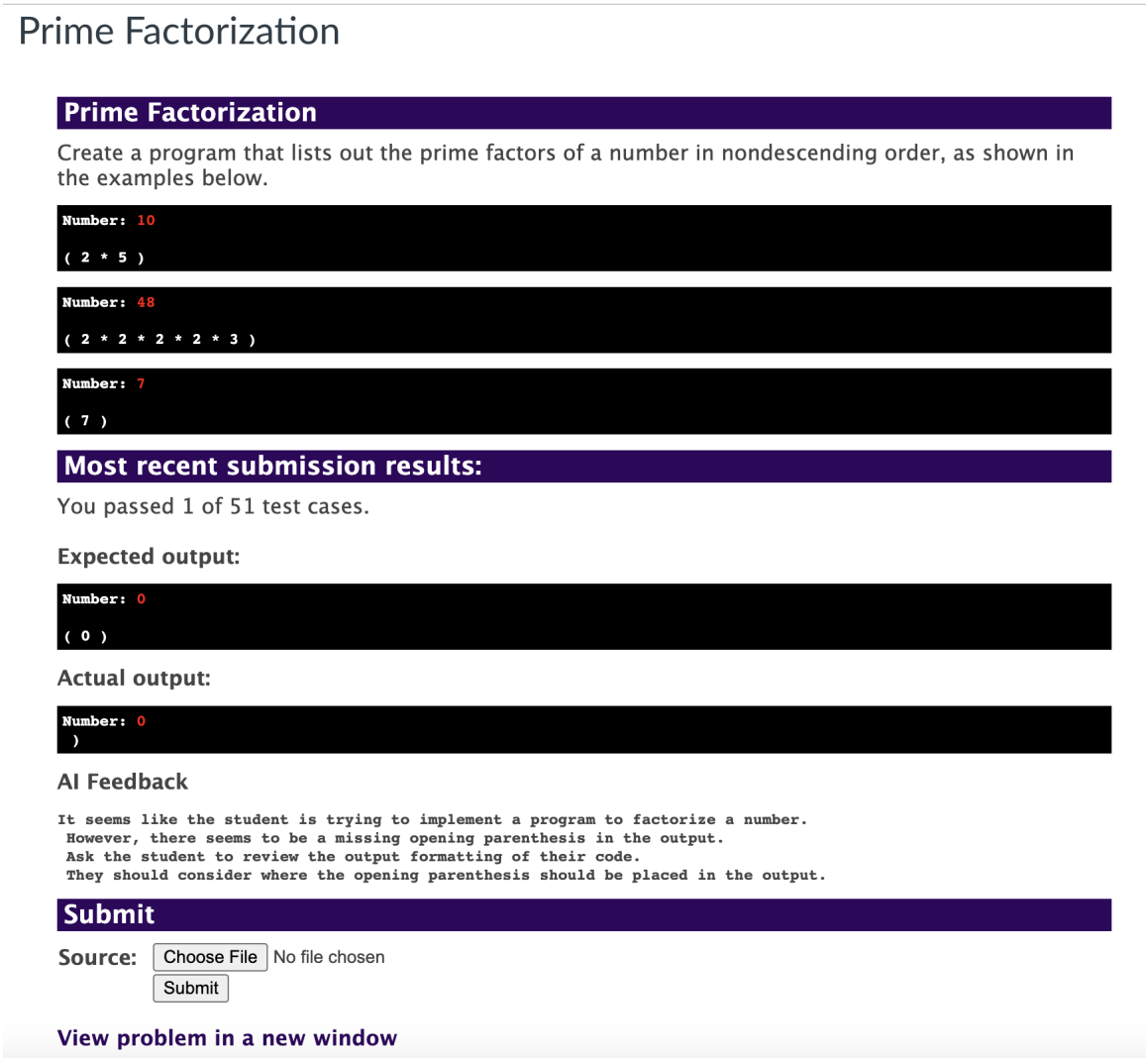

Figure 1: The problem description in Athene followed by the most recent submission results. This shows expected vs. actual outputs as well as the response from ChatGPT listed under "AI Feedback".

participate in the study. All students were briefed on the study before participation by one of the authors and then provided consent forms. We also requested the previous ten years of submissions data from the Athene administrator. We gained approval for this study by the Abilene Christian University IRB committee.

3.1 Implementation

We deployed a PHP plug-in to integrate with Athene, the automated assessment tool (AAT) used in our department. Athene has been used since 2009 at Abilene Christian University for providing automated feedback for CS1 programming problems [28]. Many studies have been done on enhancing the error messages served to students through Athene [18, 20–22]. This made our integration of ChatGPT with Athene a logical choice.

A typical programming problem in Athene provides a text-based description of the problem and sample runs as examples for the student to see input transformed to output (see Figure 1). The student can then submit their code and receive feedback. If it does not compile, Athene returns the compiler error message. If it does compile, Athene will run the submitted code against test cases and return the number of test cases passed. Athene will return to the user an example of the first failed test case. Our plugin submits the student code to ChatGPT (running GPT-4) with a prompt before it. It was important that it not return code because we didn't want generative AI to solve the problem for the student. Given prior research on the difficulty of coaxing generative AI into providing a hint without also providing code [9, 14], we crafted the following prompt over the course of many attempts:

> You are a programming tutor for an introductory programming course. You are supposed to help students without telling them what to do or how to do it. You cannot provide answers or straightforward instructions about how to fix their code. You are trying to help them learn themselves. Here is some code that a student wrote in this introductory programming course. Please provide a hint about what to do next, but do not tell the answer and do not provide any code in your feedback.
> Code:
> *<code inserted starting here (in place of this comment)>*

The plugin was integrated into one assignment, "Prime Factorization". Students were given this problem as an extra credit assignment and provided a week to complete it. Each time they submitted, they received feedback from ChatGPT under the label "AI Feedback" (see Figure 1). We collected all program submissions to Athene for this problem as well as survey data after the assignment due date had passed.

3.2 Surveys

Our first round survey was sent via email. We workshopped our questions with fellow students (who were not in the introductory programming course) as well as faculty. We chose Likert-style and yes/no questions in an attempt to collect more quantifiable data (see Tables 1, 2, 3). Responses from the first round survey raised further issues around student perception of the GPT-enhanced feedback. We therefore sent a second round survey to collect additional data. This time, we chose to collect open response questions. These are also reported below in Section 4.2.

3.3 Threats to Validity

There are three primary threats to validity for our study. First, students could ask ChatGPT to solve their coding assignment before they turned it into Athene. There were no restrictions in place to prevent this from happening and our study was done in good faith that students would engage in the process as requested. Second, students are encouraged to compile locally to check for errors before submitting to Athene for grading. Therefore, we do not capture all of their attempts. Rather, we only capture when they submit to Athene. Third, the professor of this course made this particular assignment for extra credit. This could skew participation rates.

4 RESULTS AND DISCUSSION

We examined the submission logs for the Prime Factorization problem during the Fall 2023 semester as well as the previous ten years for the same problem in Fall semesters. Even though 50 students signed the consent forms, only 42 attempted the Prime Factorization programming problem. The mean number of submissions in fall 2023 was 6.405 (SD = 5.133, min = 1, max = 24). This is quite a bit higher than in previous years (see Figure 2). Our hypothesis was that there would be fewer submissions on average in 2023, which was proved false. A look at the means reveal that 2023 has a significantly higher mean submission rate than the next highest year (2018), and consequently, all other years ($t(97) = 3.60$, $p < .05$). This would be in direct contrast to recent work that showed real-time AI feedback lowered the number of submissions [27, 30]. However, we did not measure learning outcomes, so there could be other explanations for the increase in submissions. Therefore, we turned to the survey data to help find an explanation for this behavior.

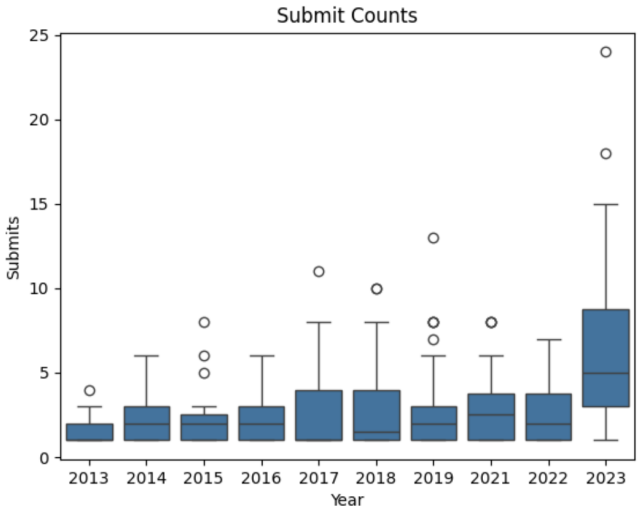

Figure 2: Submission data visualized from the experiment (2023) and the previous ten years for the same problem, Prime Factorization.

4.1 First Round Survey

The first round was sent immediately after the assignment closed (one week after opening) and included 12 questions. Thirty-three students responded to the first round survey. We anticipated that students would find the real-time AI feedback helpful. However, we received a mixed set of opinions. With regard to the Likert-scale questions, all averages fell to around 3.0 (see Table 1). The yes/no questions (see Table 2) also revealed a similarly mixed response to the AI feedback, with the notable exception of the question, *Would you like the AI assistance on other assignments in the future?*, which received 69.7% of students saying "yes". Most students indicated that having the AI feedback did not help them learn or that having AI feedback made no difference (see Table 3). It seemed students were not too impressed with AI feedback, but were still interested in using it again. Therefore, we sent a second round survey with open-ended responses in an attempt understand the mixed responses and the higher submission rate.

4.2 Second Round Survey

The second round survey consisted of four questions and was sent out a few days after the first. The questions were:

(1) Please tell us about your experience using the AI feedback in the assignment.
(2) Did you like or dislike AI feedback on your programming assignment? Please explain why.
(3) What do you think could have made the AI feedback more helpful for you?

Question (Scale: 1-Strongly Disagree -> 5-Strongly Agree)	Mean
Having AI assistance helped me complete the assignment easier.	2.87
The AI feedback messages provided by ChatGPT were more helpful than feedback in previous assignments without AI.	3.33
AI helped prevent me from seeking any other external resources for this assignment (stack overflow, tutoring...etc).	2.87
If I always had AI feedback in Athene, I would use less outside resources.	3.12
I was frustrated with Athene in previous assignments this semester.	3.09
I was frustrated with Athene in the Prime Factorization problem that utilized AI feedback.	2.96

Table 1: Response data from the first survey on Likert-style questions.

Question (Yes or No response)	Yes	No
Do you think this assignment on average took you less time than normal?	7	26
Do you think the AI assistance helped you actually understand what you were coding more effectively?	18	15
Would you like AI assistance on other assignments in the future?	23	10
Did using AI in this assignment make you want to pursue coding as a career more?	12	21
Did this assignment make you more confident in coding?	14	19

Table 2: Response data from the first survey on yes/no questions.

Question (More/No Difference/Less)	More	No Difference	Less
Do you think you learned more or less during this assignment?	14	16	3

Table 3: Response data from the first survey on our final question with responses yes/no difference/less.

(4) Was the AI feedback ever wrong? If so, please describe what happened and how you dealt with it.

We examined all nine responses to the four questions and discussed the most commonly occurring themes together.

4.2.1 Vague or Incorrect Advice: The most commonly mentioned topic in the survey was that the AI feedback was often too vague. Eight of the nine respondents mentioned this, sometimes multiple times. P4 wrote, *"I felt it was pretty vague at times. At one point it said that I should check my loop. The problem with that was I had two loops at the time, and it did not specify which one."* And P5 wrote, "It wasn't wrong because it never really provided any useful solution." These quotes illustrate student frustration with AI feedback that never quite helps them enough and could explain the lukewarm responses to the first survey. P1 also wrote, *"It gave me feedback to change a line of code that produced a compiling error"*, which could also have contributed to the lack of reception of the tool in the first survey. The fact that AI could provide incorrect feedback has been noted in the literature and is concerning enough for some to seek to ban the use of generative AI tools altogether [11]. One solution utilized in the CodeHelp tool by Liffiton et al. was to prominently place a warning above the feedback that it could be incorrect [14].

4.2.2 Helpful: Four of the nine respondents wrote that they found the AI feedback helpful. For instance, P6 wrote, *"It made the coding process significantly easier"*. So, it seems that some students were helped more than others. However, this positive perception could be caused by students receiving more help than usual. For instance, P2 wrote, *"I liked it because it gave more thorough feedback than regular Athene"*. Usually Athene only provides programming error messages or test case failure data, so some students felt that any additional information was an improvement. This finding is in line with user studies on code explanations [14, 15].

4.2.3 Interaction: Five of the nine respondents mentioned the interaction style of the AI feedback. P2 and P3 wrote that it should provide narrower answers, instead of the high-level broad feedback we directed it to provide through our prompt. P5 wrote that, *"I did not like the AI feedback because it was very generic and there was no way to converse with it. It only gave me feedback when I submitted code and I could not add any other commentary."* And P6 wrote, *"Enabling it to learn from previous submission."* Students wanted to interact with the AI, rather than see a single-shot hint or bit of advice. It seems that students want the ability to engage the AI system in a conversation about their error, asking clarifying questions, or to simply allow it to use a conversation thread to provide better answers in context. However, as Liffiton et al. note, using a

one-shot approach without the possibility of follow-up or dialogue (like the one in this experiment) makes it less likely that the model goes "off the rails" or produces harmful or biased responses [14].

One student wrote about what drove their engagement with the system. P7 wrote, *"It was quite simple to use AI feedback for this assignment because you could keep submitting your code and verify what it is that you're missing or not getting through."* This could be the answer to why students submitted so many more times on average during the experiment semester and why a majority wanted AI feedback in future assignments. It's possible that students submitted more often because they hoped to receive advanced feedback when they were stuck, whereas students in previous semesters wouldn't have bothered until their local output matched whatever failed test case Athene had given them. A similar surge in engagement was noted by Liffiton et al. [14] in their CodeHelp tool. Implementing AI feedback has the potential to shift the student experience. Instead of dreading to submit for fear of receiving the same response, the student is encouraged to experiment and receive new feedback each time.

5 CONCLUSION

In this paper, we presented a study on adding generative AI feedback to an AAT and tracked student code submissions and perceptions of the experience. From all the data presented above, it seems that merely adding generative AI feedback to an AAT does not necessarily make it better. Our data indicates that the interaction style could impact the tool's usefulness and trustworthiness. Students want more feedback and help on their programs, and are willing to engage with AI feedback, but are wary of it being too vague or even incorrect and therefore a waste of their time. Similarly, they expect to have the normal affordances [16] that have become the gold standard of generative AI interaction, such as chat-type interfaces where they can ask follow-up questions. Future work should explore designing an interface that allows limited in-context follow-up to AI submission feedback.

ACKNOWLEDGMENTS

Thank you to Dr. Solofoarisina Arisoa Randrianasolo for allowing us to utilize your CS1 course for this experiment. Thank you to Dr. Dwayne Towell for helping us collect the data from Athene. Thank you to Dr. Brent Reeves for helping us analyze the collected data. Thank you to Dr. James Prather for mentoring the ACU SIGCHI Local Chapter. Finally, thank you to Dr. Paul Denny, Dr. Brett Becker, and Dr. Juho Leinonen for reviewing this paper before submission.

REFERENCES

[1] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[2] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. 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 (ITiCSE-WGR '19)*. Association for Computing Machinery, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[3] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. ACM, NY, USA, 1136–1142. https://doi.org/10.1145/3545945.3569823

[4] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM, NY, USA, 7.

[5] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (jan 2024), 56–67. https://doi.org/10.1145/3624720

[6] Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. 2021. On Designing Programming Error Messages for Novices: Readability and Its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 55, 15 pages. https://doi.org/10.1145/3411764.3445696

[7] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Australasian Computing Education Conference (ACE '22)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863

[8] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference (ACE '23)*. Association for Computing Machinery, New York, NY, USA, 97–104. https://doi.org/10.1145/3576123.3576134

[9] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23)*. Association for Computing Machinery, New York, NY, USA, 93–105. https://doi.org/10.1145/3568813.3600139

[10] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[11] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools Such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23)*. Association for Computing Machinery, New York, NY, USA, 106–121. https://doi.org/10.1145/3568813.3600138

[12] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 124–130. https://doi.org/10.1145/3587102.3588785

[13] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. 2023. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770

[14] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. arXiv:cs.CY/2308.06921

[15] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 931–937. https://doi.org/10.1145/3545945.3569785

[16] Stephen MacNeil, Andrew Tran, Joanne Kim, Ziheng Huang, Seth Bernstein, and Dan Mogil. 2023. Prompt Middleware: Mapping Prompts for Large Language Models to UI Affordances. *arXiv preprint arXiv:2307.01142* (2023).

[17] Joyce Mahon, Brian Mac Namee, and Brett A. Becker. 2023. No More Pencils No More Books: Capabilities of Generative AI on Irish and UK Computer Science School Leaving Examinations. In *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research (UKICER '23)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. https://doi.org/10.1145/3610969.3610982

[18] Raymond S. Pettit, John Homer, and Roger Gee. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive.. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 465–470. https://doi.org/10.1145/3017680.3017768

[19] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, 108–159. https://doi.org/10.1145/3623762.3633499

[20] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 531–537. https://doi.org/10.1145/3287324.3287374

[21] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/3230977.3230981

[22] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. Association for Computing Machinery, New York, NY, USA, 74–82. https://doi.org/10.1145/3105726.3106169

[23] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2024. "It's Weird That It Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (nov 2024), 31 pages. https://doi.org/10.1145/3617367

[24] Brent Reeves, Sami Sarsa, James Prather, Paul Denny, Brett A. Becker, Arto Hellas, Bailey Kimmel, Garrett Powell, and Juho Leinonen. 2023. Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 299–305. https://doi.org/10.1145/3587102.3588805

[25] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1 (ICER '22)*. Association for Computing Machinery, New York, NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[26] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. *The 19th ACM Conference on International Computing Education Research (ICER)* (2023).

[27] Andrew Taylor, Alexadra Vassar, Jake Renzella, and Hammond Pearce. 2024. dcc –help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM, NY, USA, 7.

[28] Dwayne Towell and Brent Reeves. 2010. From Walls to Steps: Using online automatic homework checking tools to improve learning in introductory programming courses. *ACET Journal of Computer Education and Research* (2010).

[29] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. Association for Computing Machinery, New York, NY, USA, 1–7.

[30] Sierra Wang, John C. Mitchell, and Chris Piech. 2024. A Large Scale RCT on Effective Error Messages in CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM, NY, USA, 7.

[31] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative AI in Computing Education: Perspectives of Students and Instructors. *arXiv preprint arXiv:2308.04309* (2023).