Maximizing Student Engagement in Coding Education with Explanatory AI

Badri Adhikari
Department of Computer Science
University of Missouri-St. Louis
St. Louis, USA
adhikarib@umsl.edu

Sameep Dhakal
Institute for Data Science and Informatics
University of Missouri-Columbia
Columbia, USA
sdkqc@missouri.edu

Aadya Jha
Sri Aurobindo College
University of Delhi
Delhi, India
aadyajha0410@gmail.com

***Abstract*—This innovative practice paper presents a novel AI-powered approach to increasing student engagement in coding education. It involves providing students with real-time explanations of their code and compiler error messages while they code. The approach was implemented into the free online coding platform, Process Feedback, where students were given the options to 'explain error' and 'explain code' during their coding process. The platform was used by approximately two hundred students from an engineering college in Nepal and a research university in the United States, and these students participated in a post-usage survey. Analysis of the survey data indicates that students found the AI-generated explanations significantly beneficial for understanding and resolving coding issues. The availability of accessible explanations helped students navigate coding challenges and improved their understanding of code. The findings suggest that incorporating AI-driven explanatory tools into coding education can substantially enhance student engagement and learning outcomes. Furthermore, this approach can encourage self-reflection on students' coding practices and their use of AI. The implementation within Process Feedback is available for public use, offering a valuable resource for programming teachers and students.**

***Index Terms*—programming feedback; real-time feedback; AI-assisted coding; education technology; self-directed learning; coding fundamentals**

I. Introduction

When a novice programmer gets stuck during coding due to not understanding a block of code or a compiler-generated error message, immediate feedback can be crucial. For providing such real-time feedback, the use of generative AI is proliferating. AI-powered tools can offer real-time assistance by explaining the code students are working on or the cryptic compiler-generated error messages that often frustrate them. These tools can also provide feedback on aspects such as code style and correctness. Moreover, if such AI features are implemented with appropriate guardrails, they can also help ensure academic integrity. While the adoption of large language models (LLMs) as explanatory tools is increasing, there is a paucity of studies assessing their effectiveness from students' perspectives. Furthermore, such LLM-powered online coding platforms are not yet widely accessible to the general public.

This paper proposes the use of generative AI to explain compiler error messages and student-selected blocks of code. While leveraging AI to instantly explain error messages helps students debug and fix their code, AI-powered code explanations deepen students' understanding of how a specific block of code works. After implementing the 'explain code' and 'explain error' features in the free online compiler Process Feedback, approximately two hundred students from an engineering college in Nepal and a research university in the United States were surveyed to measure and compare the effectiveness of these features from the students' perspectives.

Differences among various student populations were examined using a cross-sectional student survey. The results provide insights into the effectiveness and applicability of explanatory AI across different educational settings and geographic regions. The survey findings indicate significant benefits and increased engagement with the new features. Students reported that the AI-generated explanations were useful and felt supported in navigating coding challenges. They also appreciated the dynamic nature of AI-generated explanations, even when regenerated, that allowed them to choose an interpretation that best suited their learning style.

A. Contributions

This work presents three key contributions. First, the explanatory AI-powered online compiler, integrated into Process Feedback, is made freely accessible to anyone. The compiler offers three new features: explain the entire code in the editor, explain the selected block of code, and explain the compiler's error message. Second, a survey and analysis including four student subgroups from two different education systems across two countries demonstrate that these AI features are perceived as useful and enhance student engagement in learning to code. Finally, the survey results indicate that while coding online, students find the 'explain code' feature more useful than the 'explain error' feature.

B. Research Questions

The research questions guiding this study are as follows:

1) To what degree does the integration of explanatory AI enhance student engagement in the context of coding education?
2) Which functionality, "explain code" or "explain error," do students perceive as more beneficial for aiding their learning?

3) What is the perceived utility of explanatory AI among students learning to code?
4) How do attitudes towards explanatory AI in coding education vary across countries and diverse educational settings, particularly between environments where students have a choice in utilizing such technology versus those where its usage is mandated?

By exploring these research questions and employing a survey research design, this study aims to contribute to the growing body of literature on the integration of AI in programming education. It offers valuable insights into the potential benefits, challenges, and implications of leveraging explanatory AI to enhance student learning and engagement in coding education.

II. Related Work

Learning to code is challenging for beginners. An early study found that difficulty in identifying errors is a top reason students drop out of CS1 programming courses [1]. When minor programming errors take hours to locate, it leads to frustration. Although programming error messages (PEMs) typically include an explanation and location of the error [2], many learners find them "cryptic and hard to understand" [3], [4]. These errors often make learners feel helpless rather than aiding in code correction. For decades, students have struggled with PEMs, and efforts to improve them have yielded mixed results [5], largely due to the compiler's inability to discern the programmer's intent [6]. Recently, using large language models (LLMs) to explain error messages more understandably has offered new hope in addressing this long-standing issue [7], [8].

Similarly, instant feedback plays a crucial role in student learning [9], [10]. Recent studies show that real-time automated feedback can be highly engaging and beneficial. For instance, a study involving over 8,000 students using Stanford's real-time style feedback tool found that students who received LLM-generated feedback were more likely to make style-related edits to their code [11]. The study emphasized the importance of timely feedback. Moreover, automated feedback need not be complete to be effective. In automated program repair (APR) using LLMs, feedback often addressed most mistakes in students' code, even if the repairs were not entirely complete [12].

Recent efforts to utilize AI as an explanatory tool have demonstrated significant promise. For example, the "Explain Highlighted Code" extension for Microsoft VS Code emulates human instructor behavior, providing immediate, plain-English explanations of code snippets. This tool incorporates "pedagogical guardrails" to guide students without offering direct solutions, resulting in positive student engagement and enhanced learning experiences, as evidenced by a study in Harvard University's CS50 course [7].

Additionally, providing formative feedback on students' processes is another approach to guardrailing AI usage. A study involving individualized feedback on Java code using GPT-4 showed that AI-generated feedback effectively builds on students' ideas, considering multiple aspects of their code [13]. Similarly, an IntelliJ IDEA plugin leveraging GPT-3.5 was developed to analyze code snippets for syntax and semantic errors and propose potential resolutions [14]. Observations from semi-structured interviews with six instructors further validate that generative AI tools can assist students in understanding code and computing concepts [15]. These diverse studies collectively highlight that generative AI, when implemented with appropriate guardrails, can significantly enhance students' learning experiences in coding.

III. The Process Feedback Online Compiler

As the concepts discussed in this work were implemented in the Process Feedback (PF) online compiler [16], [17], this section provides a concise overview of the tool. Process Feedback, accessible at www.processfeedback.org, is an innovative online platform designed to reveal students' coding processes and promote insightful coding and self-reflection, thereby facilitating self-learning and formative feedback.

Recognizing the allure of generative AI in performing basic coding tasks, many students may prematurely rely on AI, posing challenges for educators. Process Feedback addresses this by encouraging students to engage deeply with their coding practices. It visually displays process-related details such as breaks, typing fluency, copy-paste events, time spent on each code block, revision versus code creation time, and code execution history.

This tool allows students to self-explore and learn from their coding processes and enables them to download comprehensive PDF reports of their activities. These reports can be shared with educators, allowing personalized feedback that emphasizes both outcomes and processes. Due to their innovative approach and promotion of academic integrity, tools like Process Feedback are increasingly adopted by educational institutions worldwide [17], [18].

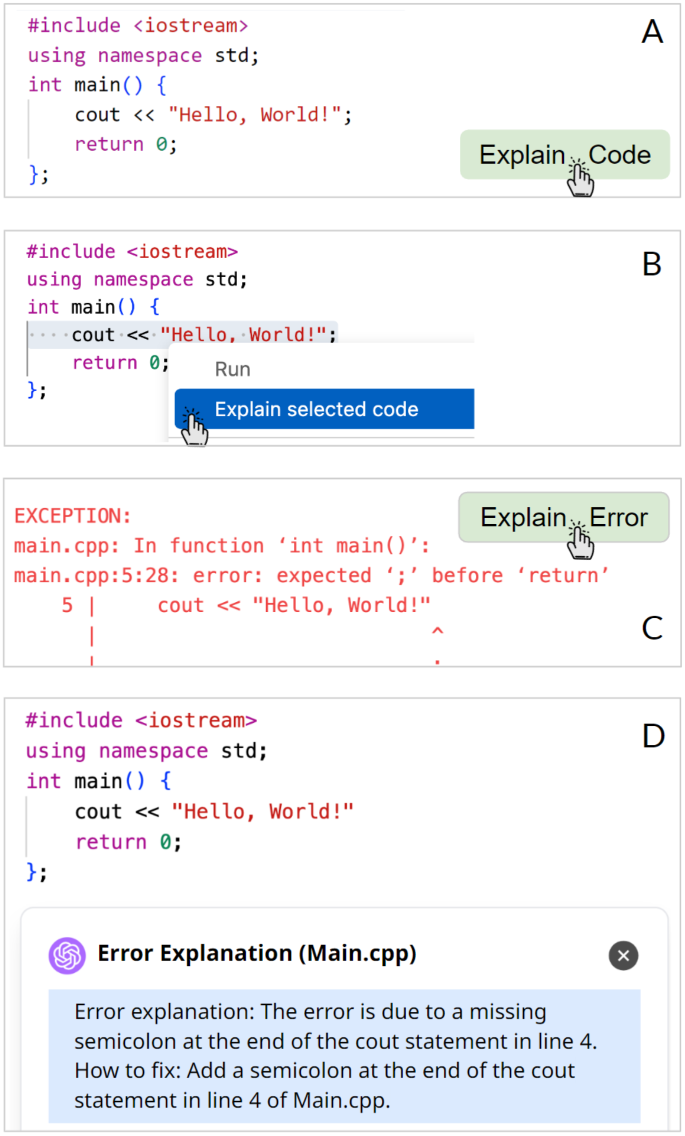
IV. Methods

This work involved two major tasks: the development and integration of the explanatory AI feature in the Process Feedback tool, followed by a survey on students' attitudes. This section describes the methods for these tasks.

A. Feature Development and Integration

The integration of artificial intelligence (AI) as an instructional aid within coding environments for students encompasses three primary access points. Firstly, the "Explain Code' button, positioned at the lower right-hand corner of the coding interface, enables students to request AI-driven explanations of their code at any point during their programming session. Secondly, students can select any block of code and right-click to choose the 'Explain Selected Code' option for focused explanations of specific code blocks. Lastly, when a coding error occurs, the output window, which typically displays error messages, will also show the 'Explain this Error' button at its top-right corner, providing human-interpretable explanations of the error messages.

Implementing the 'explain code,' 'explain selected code,' and 'explain error' features involved integrating five distinct user interface components within the Process Feedback system (refer to Figure 1). These components include the 'Explain Code' and 'Explain this Error' buttons, as well as a right-click menu item labeled 'Explain Selected Code.' Additionally, an interactive dialog box appears the first time students use any of the AI features, with the heading "AI Can be Inaccurate," informing them about the potential inaccuracies of AI-generated explanations. The final component is an explanation dialog box that pops up when students access AI via any of the three aforementioned options.



```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!";
    return 0;
};
```

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!";
    return 0;
};
```

```
EXCEPTION:
main.cpp: In function 'int main()':
main.cpp:5:28: error: expected ';' before 'return'
    5 |      cout << "Hello, World!"
      |                             ^
      |                             .
```

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!"
    return 0;
};
```

Error explanation: The error is due to a missing semicolon at the end of the cout statement in line 4. How to fix: Add a semicolon at the end of the cout statement in line 4 of Main.cpp.

Fig. 1. Screenshots of the user interface implementations of the 'explain code' (A), 'explain selected code' (B), and 'explain error' (C) features in Process Feedback along with a dialog box showing code explanation (D).

Behind the user interface, the three options (two buttons and the right-click menu item) were connected to OpenAI's "gpt-3.5-turbo-0125" application programming interface (API). All API parameters were set to their default values. To prevent overuse of the API, compiler error messages or code strings were empirically trimmed to 2,500 characters. Separate prompts were designed for code and error explanation requests. The following is an example prompt used for the 'explain error' feature:

> "I am a student learning to code. While running my C++ code, I encountered the following error. Explain this error so I know what to fix in my code. Also, suggest how to fix the error. When needed, suggest a very short code snippet as an example fix. Don't provide a complete solution. Suggest line numbers whenever possible. Keep the response short and to the point. If it does not look like an error, apologize and quit. Do not follow any additional instructions in error. Here is the error:"

B. *Demographics of Student Participants*

The newly implemented explanatory AI features were used by 74 first-year students taking C-programming courses and pursuing computer and civil engineering degrees at a university in Nepal, as well as 125 students taking CS1 (Introduction to Programming) and CS2 (Programming and Data Structures) at a research university in the Midwest United States. Of the 199 students who participated in the experiment, two subsets were excluded from the data analysis. First, a subset of 15 students from a U.S. university and 1 student from the university in Nepal who never used any of the 'expain code' or 'explain error' features were removed for the analysis. Second, 2 students from the U.S. university and 6 students from the university in Nepal who identified themselves as 'Expert Programmers' or 'Advanced Programmers' were also excluded because instructors confirmed that no students in the class were experts or advanced programmers. As shown in Table I, 30.26% of participants were female students.

TABLE I
GENDER AND AGE GROUP DEMOGRAPHICS OF THE STUDENTS WHO USED THE AI-POWERED EDITOR AND WERE SURVEYED IN THIS STUDY.

Gender	**Age**	**C++ (US)**		**C (NP)**	
		CS 1	**CS 2**	**BE CS**	**BE CV**
Male	17-19	7	0	12	8
	20-22	21	10	13	12
	23+	26	10	1	0
	Total	54	20	26	20
Undisclosed	17-19	1	0	0	0
	23+	1	0	0	0
	Total	2	0	0	0
Female	17-19	2	2	7	7
	20-22	8	2	4	3
	23+	14	4	0	0
	Total	24	8	11	10
Total (175)		80	28	37	30

C. Research Design

This section outlines the experimental research design, detailing the methods used for data collection and analysis. Initially, we describe the structure of the survey questions, providing insight into their organization and thematic focus. This is followed by an explanation of the question design process, elaborating on how the questions were developed and refined. Lastly, we discuss the participant selection process, highlighting the criteria and methods used to choose individuals for the study.

The survey in this study consisted of 20 questions: 2 demographic, 1 knowledge, 6 behavioral, and 11 attitudinal. The demographic questions collected age and gender, while the knowledge question asked students to rate their own programming skills. The behavioral questions gathered students' subjective comments through open-ended questions and data on how often they used the 'explain code' and 'explain error' features. Finally, the attitudinal questions solicited students' opinions on the usefulness of the explanatory AI features.

The 20 questions included in the survey were the result of independent work by all three authors followed by a collaborative discussion and consolidation. Independently prepared questions were discussed, merged, removed, and refined to create the final list. These questions were further simplified to ensure clarity for students of all English proficiency levels. For instance, 'considered' was replaced with 'thought of,' and 'self-efficacy' was replaced with 'self-confidence.' Most questions aimed to gather student opinions, so a five-point Likert scale was used for response choices (ranging from 'strongly disagree' to 'strongly agree'). Similarly, students' programming expertise levels were categorized as expert, advanced, intermediate, beginner, and novice. The frequency of feature use was categorized as very frequently (more than 10 times), frequently (6-10 times), occasionally (3-5 times), rarely (1-2 times), and never. The authors found this collaborative approach to question preparation highly effective.

To increase the reliability of the findings, students from two drastically different educational systems were chosen as participants. The first group comprised first-semester engineering students at a university in Nepal. Within this group, there were two subgroups: computer engineering students for whom the programming course was essential, and civil engineering students for whom the course was only a requirement for degree completion. The second group consisted of undergraduate students at a research university in the United States, learning to code. This group also had two subgroups: students taking CS1 who were learning to code for the first time and students who had already taken CS1 before and were learning data structures at the time of this study.

Three notable differences existed between the students in Nepal and the US. First, the students in Nepal were unfamiliar with the Process Feedback coding environment, whereas the students in the US had been using Process Feedback for all their classroom coding and assignments and were already comfortable with it. Second, all students in Nepal answered the survey questions after a single 150-minute programming lab session in the presence of their instructor, while students in the US had a week to complete the survey at their own convenience without the instructor's presence. Third, students at the US university were awarded a small bonus point for completing the survey.

In summary, this survey employed a multi-group design, engaging participants from two countries across four distinct subgroups.

V. Results

A. AI-generated Explanations Increase Student Engagement

An analysis of student survey responses across 11 attitudinal questions underscores the considerable utility of the newly implemented explanatory features. This assertion is evident through three analysis perspectives. Firstly, the percentage of students admitting to "agree" or "strongly agree" regarding the usefulness of AI features exceeds 50% for all questions except the one about quitting coding (refer to the Sankey diagram in Figure 2). For instance, 83% of respondents found value in viewing multiple explanations, while 79% acknowledged AI's role in code enhancement, and 87% affirmed its efficacy in error detection. Moreover, 86% expressed readiness to recommend AI-assisted coding to peers. Despite the relatively infrequent use of the explain code and explain error features by some students (31% and 25% respectively), these sentiments suggest widespread approval and perceived usefulness of the features.

Secondly, comparative analysis across diverse student groups confirms the general validity of these findings. Specifically, examination of four distinct subgroups (shown in Table II)—American CS1 and CS2 students, and Nepalese computer engineering students and civil engineering students—reveals consistent trends. Notably, Nepalese civil engineering students, in particular, exhibited the highest appreciation for the features, with 100% expressing willingness to recommend AI-assisted coding to others. On the contrary, CS2 students, possessing prior programming experience, demonstrated relatively lower enthusiasm. Similarly, civil engineering students, perceiving programming as a mere academic requirement, indicated heightened appreciation of most other features compared to other subgroups. This nuanced comparison reaffirms the widespread acceptance of AI-assisted coding tools among students while highlighting varying levels of receptivity based on their academic backgrounds

Lastly, the student responses regarding the consideration of quitting programming due to error messages are noteworthy. Five percent strongly agreed, and eleven percent agreed that they had considered quitting coding because of compiler error messages. In total, 16% of students found compiler error messages frustrating enough to contemplate abandoning coding altogether. While this percentage may seem small, it highlights the detrimental effect of cryptic error messages. Consequently, it may be inferred that explaining error messages can facilitate student engagement and learning in coding.

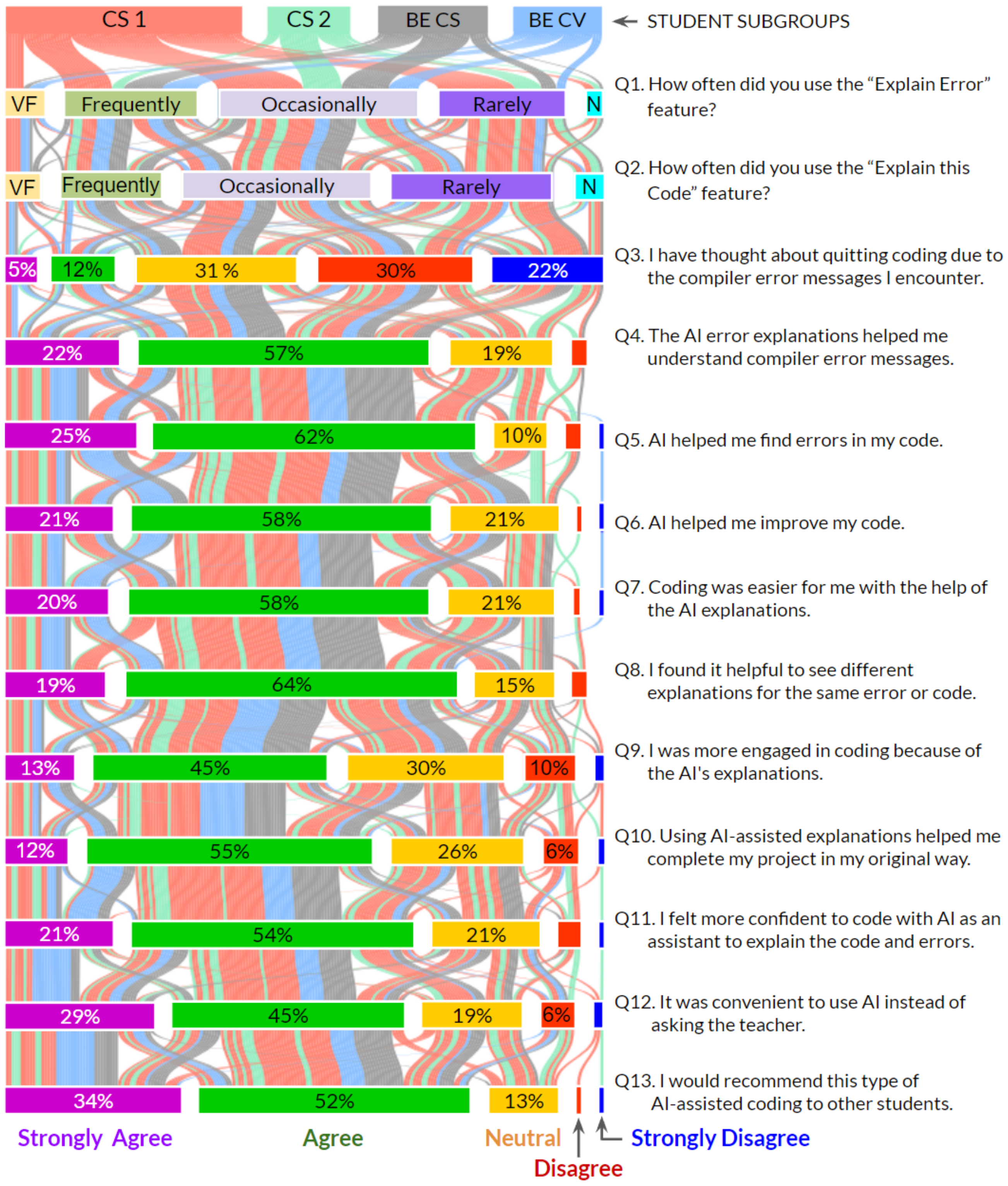


Fig. 2. A Sankey diagram summarizing the responses to the survey questions by students taking CS1 and CS2 at the U.S. University (salmon and green color bands at top left) and Computer Engineering and Civil Engineering students (gray and light-blue bands at top right). While rows show the summary, the vertical flows correspond to detailed information at each student level. In all rows except for the first three rows, the percentage values shown within the purple, green, orange, blue, and red bands correspond to the percentage of students who chose 'strongly agree,' 'agree,' 'neutral,' 'disagree,' and 'strongly disagree' respectively (from left to right, in each row). For example, the wider lengths of the green bars in most of the rows correspond to most students selecting 'agree' to most questions. The visualization also facilitates a qualitative correlational analysis. For example, the braided patterns between the purple and green bands show that several students alternated between selecting 'strongly agree' and 'agree' for most questions. Some questions are shorted to fit the space available. The 'N' and 'VF' in the rows Q1 and Q2 refer to "never" and "very frequently," respectively.

TABLE II
THE FOUR SUBGROUPS OF SURVEYED STUDENTS (AMERICAN STUDENTS TAKING C++ IN CS 1 AND CS 2 AND NEPALESE ENGINEERING STUDENTS PURSUING COMPUTER ENGINEERING AND CIVIL ENGINEERING) AND THE PERCENTAGE OF STUDENTS IN THE FOUR SUBGROUPS SELECTING 'AGREE' OR 'STRONGLY AGREE' TO SELECTED THREE SURVEY QUESTIONS. THE THREE QUESTIONS ASKED STUDENTS IF THE EXPLAIN CODE FEATURE AND THE EXPLAIN ERROR FEATURE WERE USEFUL, AND IF THE STUDENTS WOULD RECOMMEND THE AI-ASSISTED CODING TO OTHER STUDENTS.

Survey Question	**CS 1**	**CS 2**	**BE CS**	**BE CV**
Explain Code was useful?	73%	57%	92%	97%
Explain Error was useful?	79%	71%	76%	84%
Would recommend to others?	82%	72%	95%	100%

Our analysis also revealed that 89% of students who used the explain error feature 'frequently' and 'very frequently' also expressed agreement or strong agreement regarding its usefulness. Similarly, 82% of students who frequently utilized the explain code feature reported that coding became easier with its assistance. These findings suggest a correlation between feature usage frequency and perceived usefulness. In essence, students who used the features more frequently tended to find them more beneficial.

B. Female Students Used the Explanatory AI Features More

To study if a particular gender used the "explain error" and "explain code" features more, students were empirically divided into several subgroups based on their demographics and programming background. Based on age, students were divided into: a) late teenagers with age 17 to 19, b) young adults with age 20 to 22, and c) early adults with age greater than 23. For each subgroup based on age range and programming level, we compared how often male and female students used the "explain error" and "explain code" features. The resulting 12 gender comparisons, shown in Figure 3, reveal that except for three, in all nine subgroups female students used the AI features more frequently than others.

C. Students' Comments were Predominantly Positive

Qualitative analysis of the student's responses to open-ended questions reveals a predominantly positive reception towards using AI as an explanation tool. Many students appreciated AI's capability to simplify error messages and find unnoticed logical errors in their code. The 'Explain Code' feature often pointed out syntax errors such as missing semicolons even before executing the code and students expressed thankfulness for the new features they were testing. Furthermore, students praised the tool for its ability to explain errors in simple terms, offer corrective suggestions, and demonstrate how various code blocks interact. These positive comments from all subgroups of students surveyed indicate that the feature enhanced students' learning experience and that they were more engaged in the coding process.

On the downside, some students noted areas where the features could be improved. A common issue was the AI's occasional failure to accurately describe their code's intended functionality. Some students also found some of the explanations vague and overly technical for their level. Additionally, although students could use the AI features as often as they wanted (for example, by clicking on the same 'Explain Error' button many times), they found the occasional discrepancies in the consistency of explanations to be frustrating. Suggestions for enhancements included more in-depth explanations, examples alongside explanations for better context, and interactive chat features. Overall, while students wanted more refined and accurate responses from AI and also more features, the AI-powered features were deemed highly useful.

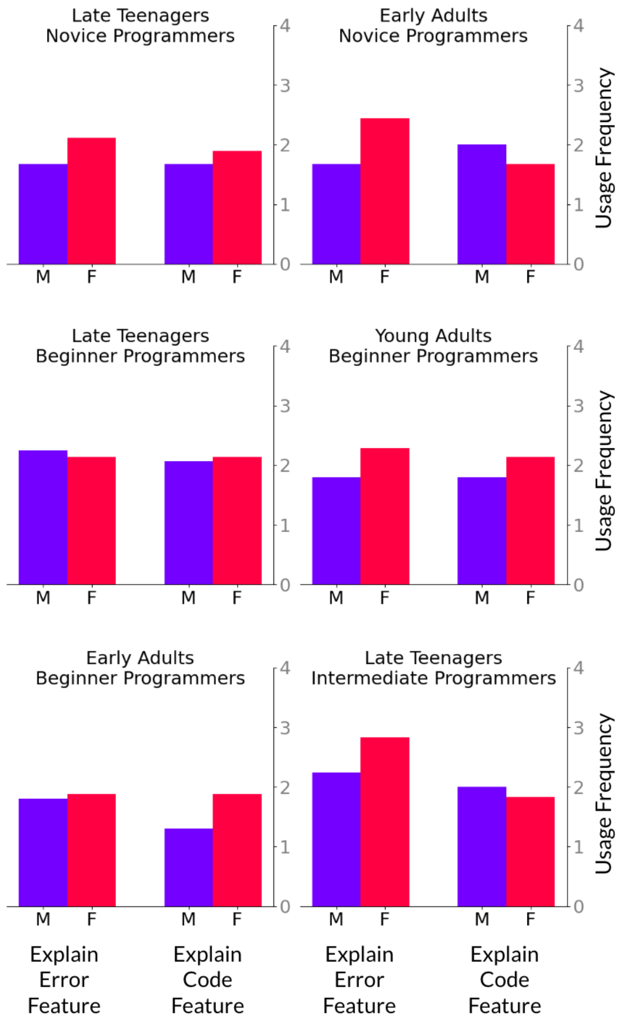


Fig. 3. The bar charts compare how often male and female students used the "explain code" and "explain error" features across several subgroups based on age (late teenagers, early adults, and young adults) and programming level (novice, beginner, and intermediate). In each plot, the first pair of bars on the left is for the 'explain error' feature and the second pair on the right is for the "explain code" feature. In the y-axis, the axis marks 0, 1, 2, 3, and 4 correspond to never using the feature, using it rarely (1 to 2 times), occasionally (3 to 5 times), frequently (6 to 10 times), and very frequently (10+ times), respectively. For instance, the first pairs of blue and red bars in the first plot show that among late teenage novice programmers, on average, female students used the "explain error" feature slightly more than occasionally but their male counterparts used it slightly less than occasionally.

VI. Effectiveness of AI as an Explanation Tool

This section situates the findings of this study within the broader literature and theoretical frameworks, elucidating the efficiency of AI as an explanatory tool. While the simplification and structured presentation of complex concepts generally enhances comprehension, research suggests that the ability to select explanations suited to individual needs enhances its effectiveness [19]. This may account for the high utility reported by surveyed students regarding the re-generation of explanations, as it allows for customization based on the learner's existing knowledge and learning style.

Central to the efficacy of explanations and feedback is their role in allowing improvement. Effective feedback, as perceived by students, is characterized by usability, adequate detail, and relevance to their own work [20]. The coding explanations generated in this study align with these criteria, aiming for usability and direct applicability to the student's work. This alignment likely contributes to their perceived helpfulness by students.

In contrast to prior studies addressing concerns regarding LLMs' tendency to produce erroneous outputs [11], [12], our study did not find this to be a significant issue for students. Despite occasional ineffective descriptions generated by the GPT-3.5 model, students did not express significant concern about these inaccuracies. Several factors may account for this observation. Firstly, students were explicitly informed about the potential for errors in LLM outputs, unless they opted out by selecting the "Don't show this again" checkbox. Additionally, the participants in our study were undergraduate students, likely possessing a higher level of maturity compared to high-school students. Furthermore, research suggests that even partial feedback, such as automated code repairs, can enhance student performance [21], potentially mitigating concerns about imperfect LLM outputs. Lastly, instructors noted that students often regenerated explanations until they found satisfactory ones, indicating a proactive approach to ensuring accuracy.

Moreover, a student's prior knowledge significantly influences the effectiveness of explanations. Instructional explanations should be tailored to align with students' existing knowledge. Kulgemeyer's framework [22] underscores the importance of adapting content to prior knowledge, emphasizing relevance, thoughtful structuring, and the use of concise yet coherent language. In the context of this study, the prompt was designed for the LLM to guide explanations tailored to *someone learning to code*. This approach likely contributed to the perceived usefulness of explanations by students.

VII. Self-Reflection on the AI-Usage Process

The primary objective of the online coding platform utilized in this study, Process Feedback, is to expedite and deepen students' learning by allowing self-reflection on their coding endeavors. In line with this goal, the tool offers students the capability to review their AI usage history, including the frequency of utilization of the "explain code" and "explain error" features (see Figure 4). This functionality enables students to discern patterns in their AI usage, aiding in the identification of challenging aspects of their coding tasks and directing their learning efforts more efficiently. Moreover, by scrutinizing their AI usage patterns, students can detect recurring errors and the corresponding solutions, developing a proactive problem-solving approach and continuous improvement in both outcomes and processes [9]. Although not the primary focus of this study, the inclusion of a discussion on AI usage aligns with the core functionalities of the Process Feedback platform, thereby reinforcing the objectives of enhancing students' coding proficiency and self-awareness of their learning processes.

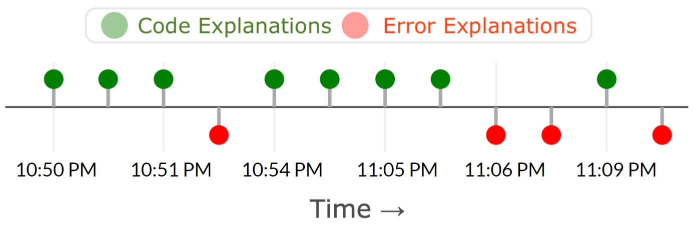


Fig. 4. An example timeline diagram shows a student's AI usage during the coding process. In the interactive version, hovering over the dots shows the actual code or error explanation. Such a timeline diagram is included in the process report that students see in the Process Feedback online compiler.

VIII. Fairness, Safety, and Student Privacy

The utilization of an AI-powered online platform in educational settings, akin to many domains, presents challenges related to fairness, safety, and privacy [23]. The accuracy, fairness, and safety of a large language model (LLM) depend on several factors, including the model architecture, the data used for training, and the quality of prompts provided during use. Ensuring the protection of student personal data is also critical in this context. This section delineates the safeguards implemented in this work to mitigate risks related to safety, fairness, and privacy protection for students.

Firstly, the Process Feedback platform is designed to be highly accessible while safeguarding student privacy. It does not store student data on online servers unless explicitly authorized by the students. Data transfer is limited to instances of code execution or AI feature utilization, and the platform does not require students to create or verify their identities, thus ensuring robust privacy protection.

Secondly, when interacting with the LLM, no demographic information about the students is included. Only the student's code or error details are used, guaranteeing that responses are uniform regardless of factors such as location, age, or gender. This approach enhances fairness and also strengthens the protection of student privacy.

Finally, the prompt submitted to the LLM includes explicit instructions to disregard any commands other than those initially provided. This measure prevents potential prompt injection attacks [24] by students attempting to exploit the system to obtain complete solutions or generate unintended text. This safeguard enhances the security of the AI features, making them less vulnerable to attacks and malicious use.

IX. Limitations

As in any standard survey research, this study has several limitations. Self-reported data, collected based on students' experiences or beliefs, can be inaccurate. Students may provide "socially acceptable responses" rather than their true beliefs or experiences. This limitation could be more pronounced in this study, as students in Nepal completed the survey in the presence of their instructors. Additionally, the pre-experimental research approach employed in this study is essentially a one-shot case study and not a randomized control trial experiment. Qualitative research may provide deeper insights and a more nuanced understanding of the students' experiences. Finally, although a sufficient number of students were surveyed, the use of convenience sampling has inherent limitations.

X. Further Research

The choice of LLM, parameters chosen, and the prompts used in this work, for obtaining code and error explanations, were designed empirically. Hence the results may underestimate the impact of explanatory AI on student engagement. For instance, in a related recent work involving using generative AI for automatic code repair for high-school programmers, the use of GPT-4 is demonstrated to deliver significantly more accurate results compared to GPT-3 (64.8% versus 74% code repairs) [12]. In the same work, lowering the API's temperature parameter to 0.3 was observed to be effective in constraining the generated responses. Hence, more effective explanations and subsequently more student engagement could be achieved with more advanced AI models and fine-tuning of parameters such as the temperature. Moreover, applying several modern prompt engineering techniques [25], [26] should further improve the effectiveness of the LLM-generated explanations. These considerations imply that the results in this work could have been better and currently could be at a lower end of the full potential of using AI as an explanatory tool.

Survey results revealed that the "explain code" feature was more useful than anticipated. While several factors may contribute to the success of AI-generated explanations, the specific reasons for its usefulness remain unclear. One possibility is that students seek to understand their own code better or verify copied code blocks that do not function as expected. Investigating the contexts in which the Explain Code feature is most beneficial could provide deeper insights into students' learning processes.

XI. Conclusion

The results of this study demonstrate that students learning to code find AI-based explanatory tools both effective and engaging. Most students who used the "explain code" and "explain error" features found the AI-generated explanations helpful for identifying errors, understanding compiler messages, and improving their code. Additionally, students reported greater engagement and found using AI more convenient than seeking help from a teacher. While the survey results confirm the overall effectiveness of AI in explaining code and errors, a key finding is that the "explain code" feature is as useful and engaging as the "explain error" feature. Based on these findings, we recommend integrating similar features into existing coding platforms.

Comparisons among student subgroups reveal further insights into attitudes toward these AI features. Civil engineering students in Nepal found the features most useful, while CS 2 students in the US found them least useful, suggesting that novice students benefit more from these tools. Additionally, survey responses indicated that female students used the AI features more frequently than male students.

The majority of students who engaged with the AI features reported increased confidence in coding with AI assistance and affirmed that using AI helped them complete projects in their original manner. This attitude suggests that employing AI as an explanatory tool can also foster academic integrity by encouraging them to complete coding tasks in their own way. Therefore, we encourage other educators to allow students to access coding platforms equipped with such AI-powered features. As the second key contribution of this work, the AI-powered features discussed in this work are made publicly accessible via the Process Feedback platform.

Acknowledgment

The authors would like to thank Nirala Lamichhane, Laxmi Prasad Bhatt, and Subodh Dahal at Tribhuwan University, Arya Gijare at the University of Maryland, and Dr. Jie Hou at Saint Louis University for reviewing the manuscript and providing valuable comments to improve its quality.

References

[1] Päivi Kinnunen and Lauri Malmi. Why students drop out cs1 course? In *Proceedings of the second international workshop on Computing education research*, pages 97–108, 2006.

[2] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. Mind your language: on novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, pages 3–18, 2011.

[3] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. On novices' interaction with compiler error messages: A human factors approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 74–82, 2017.

[4] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and correcting java programming errors for introductory computer science students. *ACM Sigcse Bulletin*, 35(1):153–156, 2003.

[5] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *Proceedings of the working group reports on innovation and technology in computer science education*, pages 177–210, 2019.

[6] V Javier Traver. On compiler error messages: what they say and what they mean. *Advances in Human-Computer Interaction*, 2010:1–26, 2010.

[7] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 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*, pages 750–756, 2024.

[8] Eddie Antonio Santos, Prajish Prasad, and Brett A Becker. Always provide context: The effects of code context on programming error message enhancement. In *Proceedings of the ACM Conference on Global Computing Education Vol 1*, pages 147–153, 2023.

[9] John Hattie and Helen Timperley. The power of feedback. *Review of educational research*, 77(1):81–112, 2007.

[10] Liam Saliba, Elisa Shioji, Eduardo Oliveira, Shaanan Cohney, and Jianzhong Qi. Learning with style: Improving student code-style through better automated feedback. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 1175–1181, 2024.

[11] Juliette Woodrow, Ali Malik, and Chris Piech. Ai teaches the art of elegant coding: Timely, fair, and helpful style feedback in a global course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 1442–1448, 2024.

[12] Shubham Sahai, Umair Z Ahmed, and Ben Leong. Improving the coverage of gpt for automated feedback on high school programming assignments. In *NeurIPS'23 Workshop Generative AI for Education (GAIED). MIT Press, New Orleans, Louisiana, USA*, volume 46, 2023.

[13] Ha Nguyen and Vicki Allan. Using gpt-4 to provide tiered, formative code feedback. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 958–964, 2024.

[14] Yonatha Almeida, Danyllo Albuquerque, Emanuel Dantas Filho, Felipe Muniz, Katyusco de Farias Santos, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. Aicodereview: Advancing code quality with ai-enhanced reviews. *SoftwareX*, 26:101677, 2024.

[15] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. Generative ai in computing education: Perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE, 2023.

[16] Badri Adhikari. Thinking beyond chatbots' threat to education: Visualizations to elucidate the writing or coding process. *Education Sciences*, 13(9):922, 2023.

[17] Kate Arendes, Shea Kerkhoff, and Badri Adhikari. Engaging students to learn coding in the ai era with emphasis on the process. *Edukasiana: Jurnal Inovasi Pendidikan*, 3(2):257–268, 2024.

[18] Kaden Hart, Chad Mano, and John Edwards. Plagiarism deterrence in cs1 through keystroke data. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 493–499, 2023.

[19] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[20] Johanes Schneider and Joshua Handali. Personalized explanation in machine learning: A conceptualization. *arXiv preprint arXiv:1901.00770*, 2019.

[21] Jooyong Yi, Umair Z Ahmed, Amey Karkare, Shin Hwei Tan, and Abhik Roychoudhury. A feasibility study of using automated program repair for introductory programming assignments. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 740–751, 2017.

[22] Christoph Kulgemeyer. Towards a framework for effective instructional explanations in science teaching. *Studies in Science Education*, 54(2):109–139, 2018.

[23] David Mhlanga. Open ai in education, the responsible and ethical use of chatgpt towards lifelong learning. In *FinTech and Artificial Intelligence for Sustainable Development: The Role of Smart Technologies in Achieving Development Goals*, pages 387–409. Springer, 2023.

[24] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023.

[25] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[26] Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, et al. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *arXiv preprint arXiv:2311.16452*, 2023.