A Large Scale RCT on Effective Error Messages in CS1

Sierra Wang
sierraw@cs.stanford.edu
Stanford University
CA, USA

John Mitchell
mitchell@cs.stanford.edu
Stanford University
CA, USA

Chris Piech
piech@cs.stanford.edu
Stanford University
CA, USA

ABSTRACT

In this paper, we evaluate the most effective error message types through a large-scale randomized controlled trial conducted in an open-access, online introductory computer science course with 8,762 students from 146 countries. We assess existing error message enhancement strategies, as well as two novel approaches of our own: (1) generating error messages using OpenAI's GPT in real time and (2) constructing error messages that incorporate the course discussion forum. By examining students' direct responses to error messages, and their behavior throughout the course, we quantitatively evaluate the immediate and longer term efficacy of different error message types. We find that students using GPT generated error messages repeat an error 23.1% less often in the subsequent attempt, and resolve an error in 34.8% fewer additional attempts, compared to students using standard error messages. We also perform an analysis across various demographics to understand any disparities in the impact of different error message types. Our results find no significant difference in the effectiveness of GPT generated error messages for students from varying socioeconomic and demographic backgrounds. Our findings underscore GPT generated error messages as the most helpful error message type, especially as a universally effective intervention across demographics.

CCS CONCEPTS

• **Social and professional topics** → **CS1**; • **Computing methodologies** → *Natural language generation*; • **General and reference** → *Cross-computing tools and techniques.*

KEYWORDS

Randomized Control Trial, Error Messages, CS1, LLM, GPT

ACM Reference Format:
Sierra Wang, John 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), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630764

1 INTRODUCTION

Compiler-generated error messages are designed to explain a programming error to the programmer; however, many error messages are confusing for students who are learning to code [7, 12]. Because error messages are frequently generated and read by novice programmers, improvements in error messaging could have widespread benefits for teaching and learning introductory programming [3, 38, 40].

Improving error messages is challenging because there is not an obvious connection between a student's misconception and the resulting compiler error; it is nontrivial to address a student's misunderstanding given the compiler's output [31]. Generative Pre-trained Transformers (GPT) offer a promising solution to this issue since they can leverage significant context and data. Alternatively, bringing students together to discuss a common error has the potential to not only improve their understanding, but also promote community in virtual learning contexts. To explore these possibilities, we developed two novel error message enhancement techniques: one with GPT generated explanations and the other facilitating collaboration on the course discussion forum.

Factors such as a student's background and educational environment also influence how they interpret an error message. These nuances make it difficult to assess whether an intervention is beneficial and have led to inconsistent results throughout the literature on error message enhancements.

To investigate the limitations of prior work, and understand the potential of GPT and collaboration based error messages, we conducted a large scale randomized controlled trial (RCT) in an introductory computer science course. We implemented six different error enhancement strategies in the course's Python IDE, and examined several metrics around the students' behavior to comprehensively assess the impact of these distinct techniques. In this paper, we present our methodology, results, and in-depth analysis considering the diverse backgrounds of learners. Our work highlights how to create more helpful error messages and, furthermore, establishes a comprehensive framework for evaluating educational tools. This approach is vital in understanding the biases of AI supported technology and developing more equitable educational interventions.

1.1 Related Work

There is considerable prior research on enhancing error messages, illustrating the opportunity to improve learning outcomes through more helpful messaging [24, 25]. Several studies have investigated what makes error messages confusing and proposed guidelines, such as readability, that would make error messages more effective [5, 15, 19, 20, 36]. There has also been significant work on enhancing error messages, such as providing simple explanations [27], including more detailed explanations [39], incorporating Stack Overflow posts [41], and more [9, 13, 17, 22].

The potential of connecting students to collaborate over shared errors is apparent. Works repeatedly mention how engineers discuss

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03…$15.00
https://doi.org/10.1145/3626252.3630764

their errors using Q/A forums [41, 43], and speculate what makes them so effective [6]. There is also substantial work on the benefits of collaborative learning [28, 34] and the downsides when there is no learning community in a course [21].

One technical challenge in providing helpful error messages has been balancing specificity and accuracy in the error message explanation; the more specific the explanation, the greater the risk of being inaccurate [31]. In the past year, advancements in Large Language Model (LLM) technology have unlocked an unprecedented ability to utilize meaningful textual context to generate accurate explanations for a specific error. Recently, others have explored the possibility of utilizing LLMs to produce error messages that show exciting potential [4, 11, 18, 29, 33, 42].

Despite numerous efforts on improving error messages, past studies yield varied results on the effectiveness of different approaches [8, 10, 14, 16, 32, 37]. Furthermore, prior research lacks analysis across varying demographics, which is necessary to understand the intervention's impact on learners from diverse backgrounds.

In our research, we aim to answer the question: **"What is the best way to improve error messages for people learning to program?"** We present our findings in this paper.

1.2 Main Contributions

(1) Two novel techniques to generate error messages for CS1 learners: one based on GPT and the other based on linking errors to relevant posts in the course discussion forum.

(2) A large-scale RCT to understand the efficacy of six different types of error messages, including our two novel techniques.

- Large-scale RCT in an open-access, online CS1 course with 8,762 students from 146 countries.
- When compared to standard error messages, students given GPT error messages repeat an error 23.1% less often in the subsequent run, and resolve an error in 34.8% fewer additional attempts.
- Only standard error messages result in significant improvement in the number of runs to resolve an error over time.
- None of the error message types affect the rate that students make errors throughout the course.

(3) A comprehensive analysis of impact of error message types across diverse learner backgrounds, finding these advantages of GPT error messages:

- No significant difference in the number of runs it takes a student to resolve their error, across Human Development Index (HDI) groups and across genders.
- For students with the least prior programming experience, significantly fewer number of runs to resolve an error compared to all other message types.

2 METHODOLOGY

We compared six error message types in a randomized controlled trial in Code in Place, an open-access, online introductory computer science course with 8,762 students from 146 countries [2, 30, 34]. We initially assigned the different error message types to the students at random, allowing students to change their error message type whenever they wished. For the integrity of the RCT, our results only include the students who used the same error message type throughout the entire course. Table 1 shows the number of students that used each error message type, and the number of errors that they generated. The descriptions and an example error message for "SyntaxError: invalid syntax," for the six error message types are:

Error Message Type	Number of Users	Number of Errors
Standard	991	86822
Long Explanation	893	72335
Forum	997	77946
Simple Explanation	918	62624
GPT - Superhero	958	53827
GPT - Default	950	53729
Total	**5707**	**407283**

Table 1: Table shows the total number of users that used each error message type, and the number of errors made of each error message type. This is data for users that used one error message type for the entire course.

Standard: default error messages from the Python runtime. We modified the raw error message by eliminating any lines referencing the compiler code, but did not add any form of enhancement. These error messages provide a baseline to compare the other enhancement strategies against. Example:

```
Traceback (most recent call last):
  File "<exec>", line 1
    def main()
              ^
SyntaxError: invalid syntax
```

Long Explanation: error messages with a detailed error explanation, often including an erroneous code example. These error messages highlight the effect of a lengthy, detailed, error message. We utilize David Pritchard's work [38, 39] to provide explanations for the most common errors, and supplement the remaining errors with Standard error messages. Example:

```
(Line 1) SyntaxError: invalid syntax
This error indicates Python was unable to interpret your
code since a grammar rule was violated. Common issues
include: strings must be enclosed within quotation
marks, conditional expressions require a colon at the
end, conditional expressions involving comparisons are
== while assignments are =, and multiplication must
always be done with *. For example,
    if x > 0 # needs colon at end of statement
Many issues can cause this error; be sure to also read
the lines before and after the indicated one.
```

Simple Explanation: error messages with a simple and concise error explanation. Comparing these with Long Explanation error messages, we analyze how much detail is helpful to provide in an explanation. We utilize Tobias Kohn's TigerPython parser [26] to

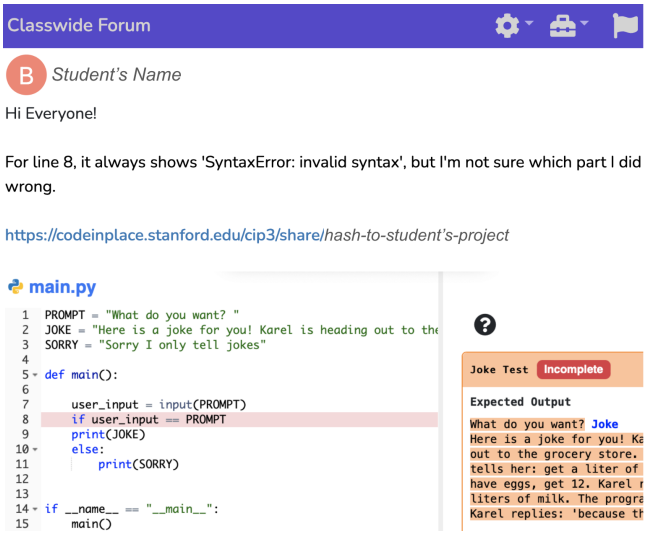

Figure 1: An example of a student asking about their error message on the course discussion forum after being directed here from the IDE.

implement error messages for syntax errors, and supplement the remaining errors with Standard error messages. Example:

```
(Line 1) SyntaxError: invalid syntax
A colon ':' is required here.
```

Forum: error messages with a link to the course discussion forum. If the forum contains a post that is relevant to this error, the error message links directly to the post and suggests that the student check it out. If no relevant post exists, the error message links to the forum and suggests that the student make a post about the error. Figure 1 shows an example post made by a student about their error. This error message type is inspired by Thiselton and Treude's work on utilizing content from Stack Overflow [41], as well as a need for collaborative learning and community in online education [21]. We implemented this error message type using Pritchard's work to detect posts about error messages [39]. We use this error message type to investigate how to connect students who are struggling with the same error, and whether it is helpful to do so. Example:

```
(Line 1) SyntaxError: invalid syntax
It looks like there is a forum post for this error,
check it out!
https://codeinplace.stanford.edu/cip3/forum?post=postId
```

GPT - Default: error message with an explanation generated by GPT. When a student receives an error, we send a prompt containing the student's code and the last line of their error message to OpenAI's Create Completion API [1]. We use the model "text-davinci-003", with a temperature of 0.1, and 1000 maximum tokens. We use a low temperature and prompt for "all the most likely reasons," in order to reduce the probability that the AI produces inaccurate explanations. This results in less confident explanations like "the reason for this error could be this, this, or this," rather than, "the reason for the error is definitely this." We analyze this novel error message type to explore the possibility of using GPT to generate error messages in real time. Example:

```
Your code had an error so we are getting a message from
a service called GPT. It might take a second, thank
you for being patient!

(Line 1) SyntaxError: invalid syntax
This error is likely caused by a missing colon (:)
after the function definition. The correct syntax for
defining a function in Python is to use the keyword
'def' followed by the function name, parentheses, and
a colon.
```

GPT - Superhero: error message with an explanation generated by GPT, in a superhero's voice. We use the same process as in GPT - Default error messages, but specifically prompt that the response be in a superhero's voice. With the latest advancements of generative AI, there is significant discussion around the use of AI in education [18]. One open question is whether AI should have a persona, and if so, in what context and what persona. Superhero error messages serve as an exploration of the efficacy of error messages in a character voice. We chose superhero because our preliminary testing demonstrated the readability and positive sentiment of this persona, which are valuable qualities in an error message [19]. Example:

```
It looks like you have an error, one minute while we
ask a superhero what is going on...

(Line 1) SyntaxError: invalid syntax
Oh no! It looks like you have encountered a SyntaxError!
This means that the code you have written is not in the
correct format. It could be that you have forgotten to
include a colon after the main() function definition,
or that you have forgotten to indent the print
statement. Fear not, for I am here to help! Just make
sure that your code is properly formatted and you
should be good to go!
```

All error messages appear in the terminal of the course's integrated development environment (IDE), shown in Figure 2 [23].

Several statistical methods were used to analyze the data. We specifically perform a one-way analysis of variance (ANOVA) to compare multiple groups simultaneously, and assess whether there are significant differences among the group means. We then apply the post-hoc Tukey test to analyze the difference between specific groups, helping to prevent Type I errors that can arise from multiple pairwise comparisons. We also perform independent t-tests to compare means between two groups.

The source code for this experiment can be found at https://github.com/sierrawang/cip-error-messages.git.

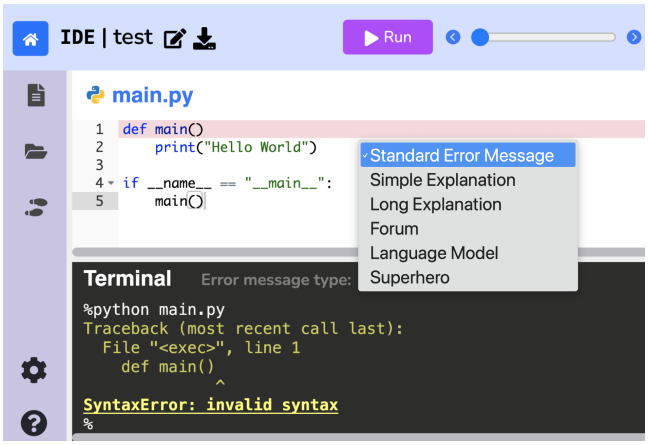

Figure 2: The IDE used for the course. All error messages appeared in the terminal. Users could change their error message type by using a dropdown list at the top of the terminal window.

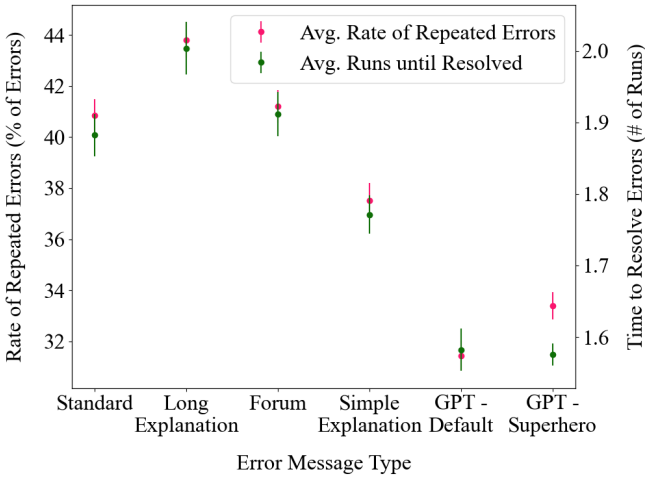

Figure 3: This figure shows the short term effects of error messages. The pink points illustrate the average rate that a student repeats the exact same error in a subsequent run. The green points illustrate the average number of runs that it takes a student to resolve their error. Both plots show that students with GPT based errors messages are able to resolve their errors fastest. Note that the y-axes start based on the lowest value for the respective data.

3 RESULTS

We examine several metrics to understand both the short and long term effects of different error message types.

3.1 Short Term Effect

To evaluate the short term effect of an error message, we investigate how effectively the message helps a student resolve their error.

First, we assess the rate that students do not resolve their error. Whenever a student generates an error, we examine whether they repeat the same error in the subsequent run. The pink data points in Figure 3 show the average percent of runs that a student receives the same error in the subsequent run, for each error message type. Students repeat their errors approximately 43.8% of runs when using Long Explanation error messages, 41.2% of runs when using Forum, 40.8% of runs when using Standard, 37.5% of runs for Simple Explanation, 33.4% of runs for GPT - Superhero, and 31.4% of runs for GPT - Default. Comparing GPT - Default (best) with Standard (baseline), and Long Explanation (worst), students repeat their error 23.1%, and 28.2%, less often in the subsequent run.

We also analyze the number of runs that it takes a student to resolve their error after receiving an error message. The green data points in Figure 3 show the average number of runs that it takes a student to resolve their error, for each error message type. It takes students the most number of runs, on average approximately 2.00 runs, when using Long Explanation error messages, 1.88 runs when using Standard, 1.91 with Forum, 1.77 for Simple Explanation, 1.58 for GPT - Superhero, and 1.58 for GPT - Default. Note that it takes at least one run to resolve an error, so these counts are lower bounded by one. Thus, it takes students 34.8% fewer additional attempts to resolve their error when using GPT - Superhero compared to Standard, and 42.6% fewer compared to Long Explanation.

In both of these capacities, the students' behavior varies significantly across the different error message types, indicating the error message's affect on the student's ability to understand and resolve their coding errors. GPT generated error messages are consistently the most effective, while Long Explanation messages are consistently the least effective. We hypothesize that GPT provides the most relevant advice on the student's cause of error, and that Long Explanation error messages, which often include an example of erroneous code that might cause the error, could often be misleading on the precise reason for the error. This suggests the importance of feedback that is tailored specifically to the code and error, and can not be hard coded based on the most common reason for the error.

In both dimensions, there is no significant difference in effect of Forum and Standard error message types. This could be because they are equally informative on what the error is, without including a text explanation of its cause.

3.2 Long Term Effect

To understand the long term effect of error messages, we examine how the rate which students generate and resolve errors changes throughout the course. These results are illustrated in Figure 4. For clarity, we graphed Standard, Simple Explanation, and GPT - Default error message types; however, our analysis is comprehensive of all error message types. Our analysis begins with Week 2 of the course because the students used a customized error message type for the first assignment, that is not included in our experiment.

The left plot in Figure 4 shows the average rate that a student generates an error each week. For each week, we calculate the number of times a student generates an error divided by the total number of times that they ran their code. We use this to determine the average error rate for a student each week, for each error message type. To account for the fluctuating difficulty of the assignments, we plot

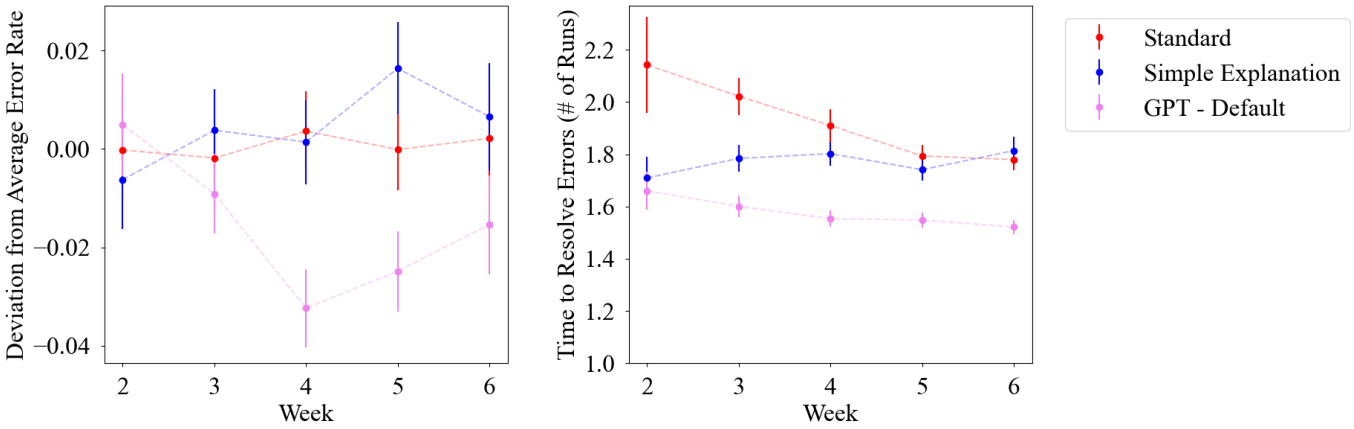

Figure 4: These plots illustrate students' behavior throughout the course. The left plot illustrates the average rate that a student generates an error each week. In order to account for fluctuating difficulty of the assignments, we look at the difference from the overall average error rate for each week. The right plot shows the average number of runs that it takes a student to resolve an error each week. These plots show that the benefits of GPT based error messages persist over time.

the difference from the overall average error rate for each week. We compare the average error rate across error message types for each week and find no consistent significant difference between the rates. In other words, we do not detect a notable difference in any error message type's influence on the rate of errors that the students made throughout the course. We do not draw conclusions about the change in error rate within a specific error message type because we hypothesize that the fluctuation in assignment difficulty significantly affects this metric.

The right plot in Figure 4 shows the average number of runs that it takes a student to resolve an error, each week. We identified the Standard message type as the sole category in which student performance significantly improved over time. This suggests that Standard error messages have the most dramatic learning curve.

Overall, these results do not highlight any specific error message type as dramatically more helpful than any other in the long term. It is possible that a five week long introductory computing course is not long enough to see significant effects in this domain. In future work, we would also like to analyze the long term effects across other dimensions, such as course enrollment and student attitude.

4 DISCUSSION

Analyzing an educational intervention across different demographics is necessary to promote equity in educational outcomes. With a course enrollment of 8,762 students from 146 countries, we are able to examine the influence of different error message types across diverse learner backgrounds. In particular, we investigate any potential bias of GPT through thorough comparison with other error message types.

Figure 5 shows the average number of runs that it takes a student to resolve their error when using each error message type, for different demographic categories. We specifically examine the results for different Human Development Index (HDI), gender, and programming experience groups.

The left plot of Figure 5 displays the average number of attempts it takes for a user to resolve their error across different HDI groups for each error message type. We find a significant difference in number of runs across the HDI categories in the case of Standard and Long Explanation error message types; these error message types are not equally effective interventions for people of different socioeconomic backgrounds. For the Simple Explanation, Forum, GPT - Superhero, and GPT - Default error message types, the differences across the HDI categories are not statistically significant.

We also analyzed the effect of error message types within each HDI category. For Low HDI, GPT generated error messages result in a significantly lower average number of runs than Long Explanation error message types. For all other HDI categories, GPT generated error messages are significantly lower than both Long Explanation and Standard error messages. We did not notice any other significant trends throughout the HDI categories. All together, these results suggest GPT generated error messages as the most helpful error message type across varying socioeconomic backgrounds.

The center plot of Figure 5 shows the average number of runs that it takes a user to resolve their error for different genders, for each error message type. In our experiment, 2,603 students identify as male, 2,953 students identify as female, and 151 students do not identify as either. We found no significant difference in efficacy across genders for any error message type. For male and female students, GPT generated error messages result in significantly fewer runs than all other error message types. For students who do not identify as male or female, GPT generated error messages result in significantly fewer runs than Long Explanation and Simple Explanation, and there is no significant difference between any other error message types. This is likely because we have less data on this demographic.

The right plot of Figure 5 shows the average number of runs that it takes a user to resolve their error when using each error message type, over varying levels of prior programming experience. For the people with the least prior programming experience, GPT generated error messages result in significantly fewer runs than

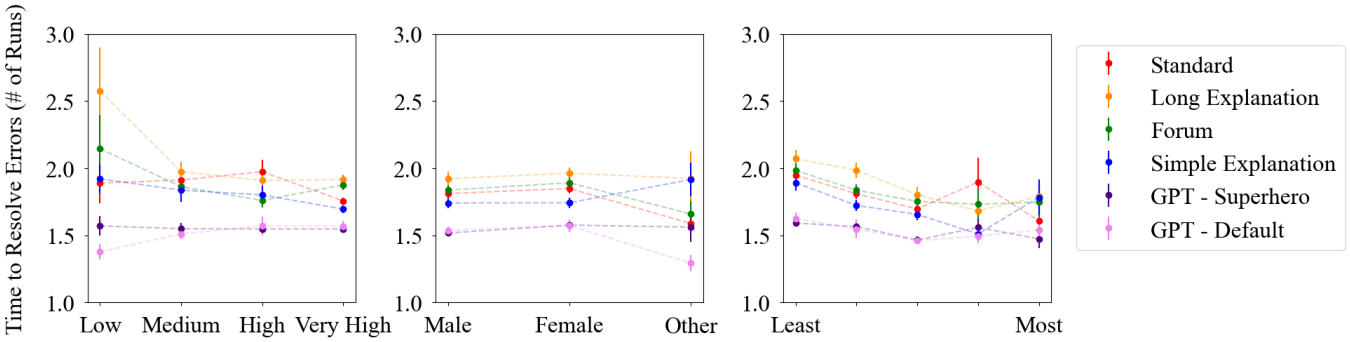

Figure 5: These plots illustrate the impact of error message type across different demographic groups. In each plot, we examine the average number of runs it takes a user to resolve their error when using different error message types. We specifically examine the results for different Human Development Index (HDI), Gender, and Programming Experience groups.

all other error message types. For the people with the most prior programming experience, there is no significant difference in runs among any of the error message types.

These findings suggest that GPT generated error messages are the most consistently helpful across various demographics compared to other error messages types.

5 LIMITATIONS

In the experimental setup and analysis, we made several decisions to ensure equitable treatment of the students and the integrity of the study. For one, we enabled students to switch between different error message types, ensuring that nobody was constrained to use an error message type that proved significantly less effective. To account for this, our analysis only includes students who used the same error message type throughout the entire course. Second, the Simple Explanation error messages were exclusively for syntax errors, and Long Explanation error messages were restricted to the errors defined by Pritchard's work [39]. We supplemented missing error messages for these types with Standard error messages. The students initially received handcrafted error messages specific to their first assignment before our error message types were introduced in the second week. Our analysis, therefore, commences from week 2. The demographics analysis was not incorporated into the RCT since there was not a consistent recruitment process across different global regions.

Our research also faced limitations associated with our data collection methods. Every time a student ran their code, we logged the code, the error produced by pyodide, the error message shown to the student, the error message type used, and a timestamp generated in the front-end. When evaluating the effectiveness of the error messages, we assess both the rate that students resolve errors upon receiving an error message and the number of attempts it takes to resolve the error. We refrain from analyzing the effect in time units due to the unreliability of timestamp precision. Previous research indicates that the number of runs serves as a comparable measure to time in this context [35]. When comparing error messages, we parsed the messages to mitigate minor differences (such as function names), but this process was not flawless. We do not believe that this influences the results since the overwhelming majority of error messages are among those we could cleanly parse. We also do not assess the influence of error message types on student satisfaction and performance in the course, posing areas for future research.

Finally, there is future work in further analyzing GPT's potential, as well as its limitations and bias. We utilized OpenAI's text-davinci-003 to generate GPT-based error messages, chosen for its efficient speed and cost-effectiveness. Still, there was a delay in GPT generated errors that may have inadvertently affected student interpretations, adding a confounding factor in our analysis. Although we have substantial evidence that the benefits of GPT generated errors were seen across various demographics, our analysis could be improved with a meaningful sentiment analysis of error messages in relation to different demographics. This is necessary to determine whether the AI is detecting different demographics through their code and whether it provides specific feedback based on this information. As generative AI technology continues to advance, ongoing studies of this nature are crucial to ensure we develop educational tools that are inclusive and beneficial for all.

6 CONCLUSION

Our research addresses the opportunity to improve learning outcomes in computing education through more helpful error messages. Through a large-scale randomized control trial, we evaluate the efficacy of six distinct error message enhancement strategies. We also present novel approaches for generating error messages with GPT in real time and constructing error messages that facilitate communication on the course discussion forum. Our results provide compelling evidence on the helpfulness of error messages generated by GPT, particularly for students of varying demographics. Our methodology also serves as a useful framework for analyzing the efficacy of error messages and ensuring that future innovations promote equity in educational outcomes.

7 ACKNOWLEDGEMENTS

We would like to thank the Carina Foundation and all the wonderful people of Code in Place for making this work possible.

REFERENCES

[1] 2021. Create Completions - API Reference. https://platform.openai.com/docs/api-reference/completions/create. Accessed: 2023-08-17.

[2] 2023. Code in Place. https://codeinplace.stanford.edu/ [Online; accessed August-2023].

[3] Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM technical symposium on computer science education*. 522–527.

[4] Rishabh Balse, Bharath Valaboju, Shreya Singhal, Jayakrishnan Madathil Warriem, and Prajish Prasad. 2023. Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 292–298.

[5] Titus Barik. 2018. *Error messages as rational reconstructions*. North Carolina State University.

[6] Titus Barik, Denae Ford, Emerson Murphy-Hill, and Chris Parnin. 2018. How should compilers explain problems to developers?. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 633–643.

[7] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do developers read compiler error messages?. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 575–585.

[8] Brett A Becker. 2015. An exploration of the effects of enhanced compiler error messages for computer programming novices. (2015).

[9] Brett A Becker. 2016. An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 126–131.

[10] Brett A Becker. 2016. A new metric to quantify repeated compiler errors for novice programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 296–301.

[11] 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*. 500–506.

[12] 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. 2019. 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* (2019), 177–210.

[13] Brett A Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (2016), 148–175.

[14] Brett A Becker, Kyle Goslin, and Graham Glanville. 2018. The effects of enhanced compiler error messages on a syntax error debugging test. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 640–645.

[15] Brett A Becker and Catherine Mooney. 2016. Categorizing compiler error messages with principal component analysis. In *12th China-Europe International Symposium on Software Engineering Education (CEISEE 2016), Shenyang, China*. 28–29.

[16] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. 273–278.

[17] Paul Denny, James Prather, and Brett A Becker. 2020. Error message readability and novice debugging performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 480–486.

[18] 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. 2023. Computing Education in the Era of Generative AI. *arXiv preprint arXiv:2306.02608* (2023).

[19] 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*. 1–15.

[20] Tao Dong and Kandarp Khandwala. 2019. The Impact of" Cosmetic" Changes on the Usability of Error Messages. In *Extended abstracts of the 2019 chi conference on human factors in computing systems*. 1–6.

[21] Joselyn Goopio and Catherine Cheung. 2021. The MOOC dropout phenomenon and retention strategies. *Journal of Teaching in Travel & Tourism* 21, 2 (2021), 177–197.

[22] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R Klemmer. 2010. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1019–1028.

[23] Thomas Jefferson, Chris Gregg, and Chris Piech. 2024. PyodideU: Unlocking Python Entirely in a Browser for CS1. In *Proceedings of the 55th acm technical symposium on computer science education*. in press.

[24] Tobias Kohn. 2017. *Teaching Python programming to novices: Addressing misconceptions and creating a development environment*. ETH Zurich.

[25] Tobias Kohn. 2019. The error behind the message: Finding the cause of error messages in python. In *Proceedings of the 50th ACM technical symposium on computer science education*. 524–530.

[26] Tobias Kohn. 2021. Tobias-Kohn/TigerPython-Parser. https://github.com/Tobias-Kohn/TigerPython-Parser/ [Online; accessed August-2023].

[27] Tobias Kohn and Bill Manaris. 2020. Tell Me What's Wrong: A Python IDE with Error Messages. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1054–1060.

[28] Marjan Laal and Seyed Mohammad Ghodsi. 2012. Benefits of collaborative learning. *Procedia-social and behavioral sciences* 31 (2012), 486–490.

[29] 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*. 563–569.

[30] Ali Malik, Juliette Woodrow, Brahm Capoor, Thomas Jefferson, Miranda Li, Sierra Wang, Patricia Wei, Dora Demszky, Jennifer Langer-Osuna, Julie Zelenski, Mehran Sahami, and Chris Piech. 2023. Code in Place 2023: Understanding learning and teaching at scale through a massive global classroom. https://piechlab.stanford.edu/assets/papers/codeinplace2023.pdf.

[31] Davin McCall and Michael Kölling. 2014. Meaningful categorisation of novice programmer errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, 1–8.

[32] 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*. 465–470.

[33] Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. *arXiv preprint arXiv:2302.04662* (2023).

[34] Christopher Piech, Ali Malik, Kylie Jue, and Mehran Sahami. 2021. Code in place: Online section leading for scalable human-centered learning. In *Proceedings of the 52nd acm technical symposium on computer science education*. 973–979.

[35] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 153–160.

[36] James Prather, Paul Denny, Brett A Becker, Robert Nix, Brent N Reeves, Arisoa S Randrianasolo, and Garrett Powell. 2023. First Steps Towards Predicting the Readability of Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 549–555.

[37] 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*. 74–82.

[38] David Pritchard. 2015. Frequency distribution of error messages. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*. 1–8.

[39] David Pritchard. 2020. cemc/cscircles-wp-content. https://github.com/cemc/cscircles-wp-content/blob/master/plugins/pybox/plugin-errorhint-en_US.php [Online; accessed August-2023].

[40] Rebecca Smith and Scott Rixner. 2019. The error landscape: Characterizing the mistakes of novice programmers. In *Proceedings of the 50th ACM technical symposium on computer science education*. 538–544.

[41] Emillie Thiselton and Christoph Treude. 2019. Enhancing python compiler error messages via stack. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[42] Patricia Widjojo and Christoph Treude. 2023. Addressing Compiler Errors: Stack Overflow or Large Language Models? *arXiv preprint arXiv:2307.10793* (2023).

[43] Alexander William Wong, Amir Salimi, Shaiful Chowdhury, and Abram Hindle. 2019. Syntax and Stack Overflow: A methodology for extracting a corpus of syntax errors and fixes. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 318–322.