Improving Students' Code Comprehension Ability Through LLM-Generated Practice Problems

William Jordan Kwako
wkwakow@gmail.com

1 INTRODUCTION

Students struggle to meet learning objectives in introductory computer science (CS) classes. In particular, students struggle to read code. Denny et al. (2008) discovered that students perform better on writing problems than code tracing problems. Reading code is often not directly taught in CS classes, even though the ability to read code precedes the ability to write it (Lister et al., 2009). Although there is a strong connection between learning to read and write code (2009), Simon et al. (2009) found that students' code reading and writing skills are roughly the same, but that top performers have stronger reading than writing skills, whereas others' reading skills lag behind their writing skills.

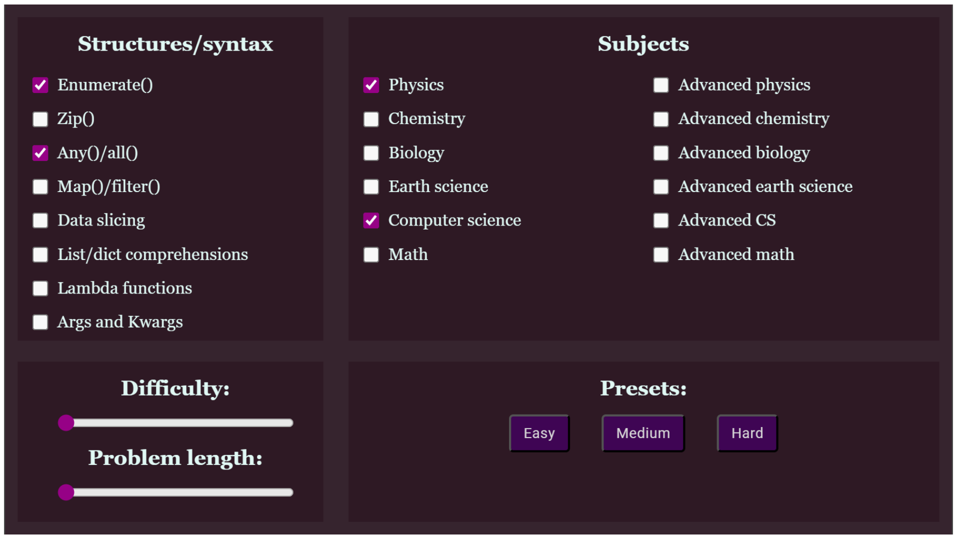
To address these issues, I developed a webapp that aims to improve students' code reading abilities by generating practice problems for students to solve. Rather than asking students to write code themselves, my project creates problems that rely on a student's code comprehension ability to solve. This instructional design choice is supported by evidence from Lister et al. (2009), who recommend that instructors incorporate activities involving reading and tracing code into computer science curricula. With respect to my investigation, I address the following research questions:

1. How do students perceive the usefulness and usability of the webapp for supporting their learning in introductory computer science courses?
2. Does regular use of the webapp improve students' code comprehension skills relative to their code writing skills?

2 METHODS

2.1 Web app development

The website was created with HTML, CSS, Javascript, and Python, and models by OpenAI and Anthropic are used to generate problems of varying types. An options panel gives users agency over several aspects of problem generation including syntax, topic, and problem length, which can be seen in Figure 1. API calls are made to OpenAI's GPT 4.1-mini model and Anthropic's Claude 3.5 Haiku model for problem generation. Before code is shown to users, I verify that it is safe to run, and use the ast python package to confirm it contains user-selected structures. Specifically, OpenAI's model is used for initial problem generation, and Anthropic's model is used to regenerate the problem should it fail to meet user specifications. Please see the Appendix for an example query.



Structures/syntax:
- [✓ Enumerate()]
- Zip()
- [✓ Any()/all()]
- Map()/filter()
- Data slicing
- List/dict comprehensions
- Lambda functions
- Args and Kwargs

Subjects:
- [✓ Physics]
- Chemistry
- Biology
- Earth science
- [✓ Computer science]
- Math
- Advanced physics
- Advanced chemistry
- Advanced biology
- Advanced earth science
- Advanced CS
- Advanced math

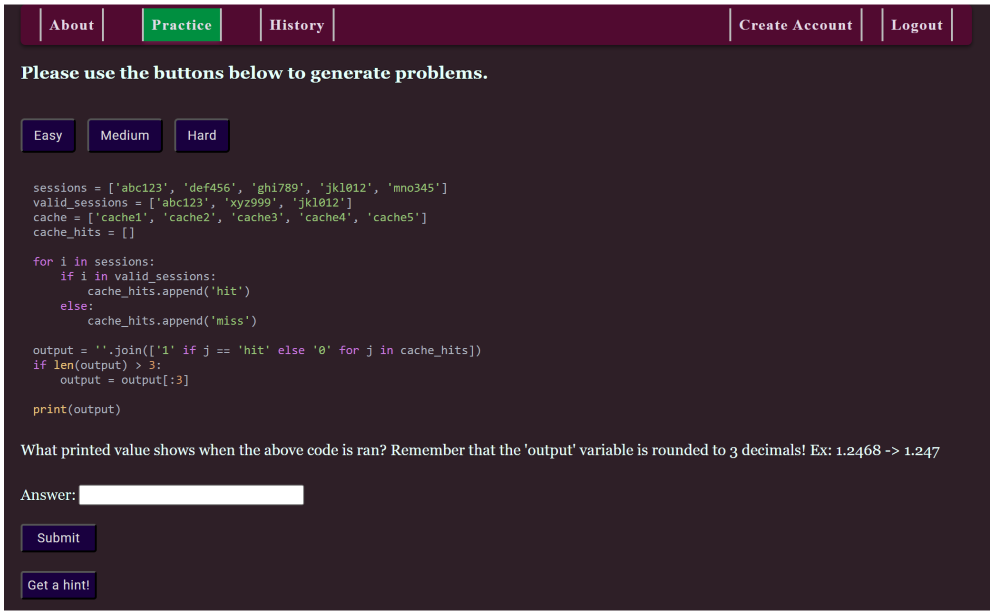
Figure 1—An image of the Generation Options panel available to users. Structures, subjects, difficulty, and problem length can be adjusted.

2.2 Item design

This tool includes three types of problems. In one, users are asked to determine what output is produced when the code runs; in the second, users are asked to write docstrings to complete class and function documentation; in the third, users must rearrange a block of code, line by line, to run without errors. In Figure 2, an image of the website's practice page is displayed, along with a generated problem.

2.3 Survey administration

A pilot survey was created that gathered feedback on the website from computer scientists and professionals. Each participant was asked to spend 10-15 minutes using the website, noting both helpful features and areas of improvement. Participants were asked about the UI, problem diversity, the feedback system, and its use for practicing code comprehension. 17 participants responded to the survey.



Please use the buttons below to generate problems.

```
sessions = ['abc123', 'def456', 'ghi789', 'jkl012', 'mno345']
valid_sessions = ['abc123', 'xyz999', 'jkl012']
cache = ['cache1', 'cache2', 'cache3', 'cache4', 'cache5']
cache_hits = []

for i in sessions:
    if i in valid_sessions:
        cache_hits.append('hit')
    else:
        cache_hits.append('miss')

output = ''.join(['1' if j == 'hit' else '0' for j in cache_hits])
if len(output) > 3:
    output = output[:3]

print(output)
```

What printed value shows when the above code is ran? Remember that the 'output' variable is rounded to 3 decimals! Ex: 1.2468 -> 1.247

Answer:

Figure 2—A picture of the ’practice’ page on the website. A problem has been generated, which requires the user to determine the output to solve.

3 RESULTS

Preliminary findings show that most participants agreed the tool produced code that could have been written by humans, and that it could improve their ability to read code. Table 1, for instance, shows responses to two related Likert-scale questions. 88% of participants agreed or strongly agreed that the tool generated human-like code, while 70% of participants agreed or strongly agreed that the tool could improve their ability to read code. Other survey questions asked about layout, hints, difficulty level, and problem variety. I have since made modifications to the tool based on this feedback.

4 FUTURE WORK

The pilot study shows promising results, and I plan to implement the webapp in an introductory computer science class, performing analyses on student submitted assignments to determine its efficacy.

5 IMPLICATIONS

Tools that improve students’ ability to read and understand code are scarce in CS education, even though there are many studies demonstrating the effectiveness

Table 1—Left: 88% of participants agreed or strongly agreed the code written by ChatGPT was human like. Right: 70% of participants agreed or strongly agreed that this website could help improve their ability to read code. No participants disagreed or strongly disagreed with either statement.

The code generated by ChatGPT was human like	**# responses**
Strongly disagree	0
Disagree	0
Neutral	2
Agree	12
Strongly agree	3

The website could help me improve my ability to read code	**# responses**
Strongly disagree	0
Disagree	0
Neutral	5
Agree	9
Strongly agree	3

of reading and tracing code. Starting a job at any medium to large software company entails getting up to speed with the code base, which could be thousands to hundreds of thousands of lines long. This webapp aims to help fill this educational gap by generating problems via LLMs for students to solve. With positive results from a quantitative study, we could add to the growing body of work demonstrating that reading code is an effective method for learning computer science.

6 REFERENCES

[1] Denny, P., Luxton-Reilly, A., and Simon, B. (Sept. 2008). "Evaluating a new exam question: Parsons problems". In: *Proceedings of the Fourth international Workshop on Computing Education Research*. ICER '08. ACM. DOI: 10.1145/1404520.1404532. URL: http://dx.doi.org/10.1145/1404520.1404532.

[2] Lister, R., Fidge, C., and Teague, D. (July 2009). "Further evidence of a relationship between explaining, tracing and writing skills in introductory programming". In: *ACM SIGCSE Bulletin* 41.3, pp. 161–165. ISSN: 0097-8418. DOI: 10.1145/1595496.1562930. URL: http://dx.doi.org/10.1145/1595496.1562930.

[3] Simon, Lopez, M., Sutton, K., and Clear, T. (2009). "Surely We Must Learn to Read before We Learn to Write!" In: *Proceedings of the Eleventh Australasian Computing Education Conference*. ACE 2009. ACS.

7 APPENDIX

A full query example: "I'd like you to generate a snippet of Python code for me. The purpose of the code is educational and should help students practice reading code. All of the requirements on this list must be met:

-Use i, j, and k for iterators in "for" loops.

-Store the output of the code in a variable called "output", where the last line prints the output variable.

-The number of iterations through the most nested for loop should never be more than 5 (if using for loops).

-The output should be limited to 10 characters or fewer, and not more than one line long. The output must be something the user can type into a text box (not contain weird characters or symbols).

-The output of the code must be easy to determine with mental math.Most code should be wrapped in functions, but that "output" variable must be outside functions and still print something at the end.

-Do not include descriptions of the code.

-Do not use input() or random() functions in the code.

-Do not include descriptions of, or introductions to, the code. Just respond with the code.

-Do not include comments, annotations, or docstrings in the code.

-Do not use ANY of these structures in the code: enumerate(), zip(), any()/all(), map()/filter(), data slicing, comprehensions, lambda functions, args and kwargs.

- Generate code from the domain of ratios and proportions related to removing or replacing parts of data.

- The length of the problem should be between 25 and 39 lines."