Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025
ASEE
Paper ID #46672

Investigating the capabilities and limitations of ChatGPT to perform programming assignments from an introductory R programming course

Dr. Lucie Guertault, North Carolina State University at Raleigh

Investigating the capabilities and limitations of ChatGPT to perform programming assignments from an introductory R programming course

1. Introduction

Large language models (LLMs) are generative artificial intelligence (AI) tools capable of performing various natural language processing tasks such as generating text and engaging in conversations with users [1]. Recent advancements in LLMs have enabled them to generate computer code in response to user prompts [2]. In addition to generating code, these tools can also assist with explaining, commenting, translating, debugging, and optimizing code [3], [4]. LLM-based tools are transforming programming education by providing real-time assistance and personalized feedback to students. However, they also present challenges, including the risk of students becoming overly reliant on AI to write code, which may hinder their understanding of fundamental programming concepts, as well as their critical thinking and problem-solving skills [5], [6], [7]. When using AI to generate code, students must still be able to interpret the outputs and assess their accuracy. Another concern related to LLM-based coding tools is content ownership and plagiarism [5].

To address these challenges, programming instructors must adapt their courses to help students leverage the benefits of LLMs while minimizing their misuse as a cheating aid and mitigating their potential negative effects on learning. A first step to adapt programming courses is to assess the effectiveness of LLM tools for generating code for course assignments. Several studies have evaluated the use of LLM-based tools in computer science courses. Ouh et al. [6] tested ChatGPT 3.5 and 4 with instructions for 80 coding exercises from their undergraduate Java programming course and evaluated the solutions generated by each version of ChatGPT. They found that ChatGPT performed well when instructions were clear and straightforward but that it sometimes produced incorrect solutions for more complex problems, particularly those involving object-oriented programming concepts. They also observed that ChatGPT 4 was more effective than ChatGPT 3.5. Similarly, Piccolo et al. [8] tested ChatGPT with 184 Python coding exercises from their introductory bioinformatics course. In their study, ChatGPT directly produced correct code for approximately 75% of the exercises. When they provided feedback to ChatGPT through follow-up prompts, its accuracy improved to 97%. ChatGPT performed better when the exercise instructions were shorter (2,000 characters vs. 9,000 characters for incorrect solutions) and when the generated code was more concise. Geng et al. [9] examined ChatGPT's performance on 31 homework assignments and exam instructions from their introductory Functional Language OCaml course, which included both programming tasks and conceptual questions. ChatGPT initially produced correct code for 52% of the homework exercises. In the midterm exam, it scored 49% on conceptual questions and 100% on coding tasks, while in the final exam, it achieved 34% on conceptual questions and 84% on coding tasks. When they used follow-up prompts that either rephrased instructions, provided hints, or supplied test cases, the quality of the responses provided by the AI tool improved. With follow-up prompts, ChatGPT's midterm exam scores increased to 62% on conceptual questions and 75% on coding tasks, while final exam scores rose to 61% and 94%, respectively. Additionally, Geng et al. [9] found that rephrasing questions and hints were the most effective prompting strategies for improving AI-generated code. Savelka et al. [10] tested text-davinci-003, another generative AI tool developed by OpenAI for structured tasks like code generation, using 599 assessments from three introductory and intermediate Python programming courses. In the introductory course, the tool scored 85.7% on multiple-choice questions and 56.1% on programming tasks. In the intermediate course, it scored 69% on multiple-choice questions and 68% on programming tasks. Popovici [11] tested ChatGPT with 72 coding exercises from their second-year Functional Programming course. They found that ChatGPT's initial responses were correct 68% of the time and improved to 86% after users provided feedback through follow-up prompts. Most of the previous studies have focused on computer science courses, with less attention given to discipline-specific programming courses that incorporate problems from a particular field. Furthermore, the coding experiments with generative AI tools were typically conducted by academic faculty members with advanced programming skills and presumably greater proficiency with generative AI than the average student enrolled in their courses.

This study aims to address some of these gaps by focusing on programming assignments from an introductory biological engineering programming course. Additionally, the participants were undergraduate students who had taken the course the previous year and had limited programming experience beyond the course, as well as no background in using generative AI to produce code. The objectives of this study are:
i) To assess the ability of a student-prompted generative AI tool, ChatGPT, to produce R code for assignments in a biological engineering programming course,
ii) To document the time, number and type of prompts required for students to be satisfied with the AI-generated code.

2. Methods

Course Description

The assignments evaluated in this study come from a 2-credit hour undergraduate course taught to Biological Engineering majors, typically in their second year. The first half of the semester focuses on Excel and the second half focuses on R programming. R programming topics that are covered include: data types and structures (vector, matrix, data frame), base arithmetic, statistical and data analysis functions, for and which programming loops, conditional tests and if / else statements, data import and export, base plots, packages and user-defined functions. The course was designed specifically for Biological Engineering majors and contextualizes instruction with assignments written in the form of real-world problems that require a solution in the form of a program. For each module, students complete approximately five programming assignments in a to gradually build their programming skills: the first assignment is a follow-along demonstration by the instructor, the second assignment is a guided group programming exercise, the next two assignments are performed during a laboratory session and the last assignment is an independent homework.

Data Collection

This study was conducted in Fall 2024. Three undergraduate students who had completed the introductory course the previous year were recruited to use ChatGPT to perform the course assignments. We will refer to them as subjects 1, 2 and 3. They had limited programming experience beyond the course and no prior knowledge of how to use generative AI to write code. They did not receive any formal training on generative AI tools. They were added to the course's learning management system for the Fall 2024 semester and were given access to all course materials, as if they were regular students in the course. Each week, they were tasked with entering the descriptions of the laboratory and homework assignments into ChatGPT and prompt it to generate code to answer the assignment. In total, they worked on thirteen assignments from five different modules:

- M8: R Fundamentals (computations with vectors) (2 assignments)
- M9: R Vectors (analysis of vectors) (2 assignments)
- M10: R Data Structures (creation, analysis, and computations with matrices and data frames) (3 assignments)
- M11: R For and Which Loops (3 assignments)
- M12: R Conditional Statements (logical tests, if/else statements combined with loops) (3 assignments)

The instructions for these assignments ranged from 815 to 2430 characters, which is on the shorter side compared to the assignments evaluated by Piccolo et al. [8]. At the time the assignments were performed, the course had not yet covered how to import external data files into R. As a result, 7 out of the 13 assignments included a starting R script that declared vector objects with the input data (Figure 1). Six of the thirteen assignment descriptions included mathematical formulas, and two assignments featured tables (3 rows × 2 columns and 2 rows × 7 columns) with input data.

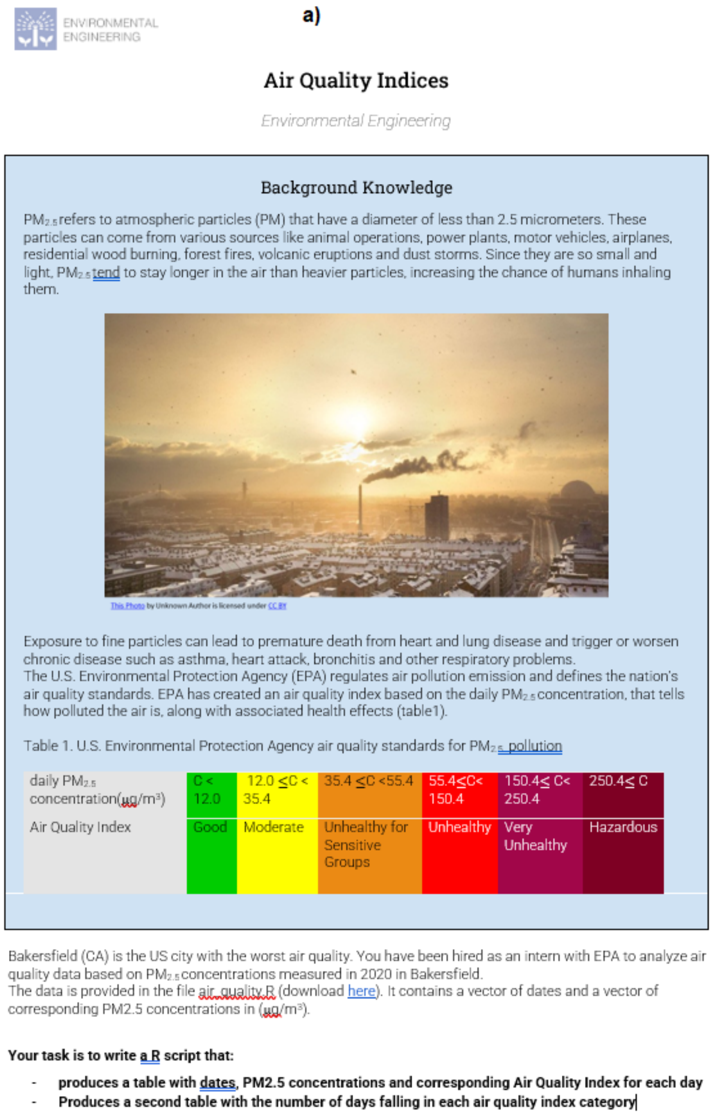
The subjects were allowed to update the assignment descriptions when pasting them into ChatGPT, particularly if tables or equations did not copy correctly. They could also paste or upload any data files provided with the assignment instructions to ChatGPT. They were allowed to prompt ChatGPT as many times as needed to update the generated code. While they were allowed to fix small syntax errors manually, they were not permitted to manually make large changes to the code. Additionally, they were allowed to run the code to check if it worked as expected. Once they were satisfied with the code, the subjects submitted their ChatGPT-generated code for grading by the course teaching assistants. These assignments were graded alongside those of the students enrolled in the course using a specific rubric. The teaching assistants were unaware of this study to avoid bias in grading. The grading criteria for each assignment included: definition of R variables for all inputs, correctness of numerical or table outputs, efficient programming (e.g., no hardcoding, reusability for other inputs), code documentation and readability (organization and comments). In addition to submitting the ChatGPT-generated code, the subjects documented their work with ChatGPT by providing a report. The report included their conversation with ChatGPT, allowing us to track the prompts they supplied and the different versions of code they received. The subjects also reported how long it took them to complete the assignment and answered the following Likert-scale question: How complicated was it to prompt ChatGPT to obtain the desired output code? (1. Not at all, 2. Slightly, 3. Moderately, 4. Very, 5. Extremely). The subjects were encouraged to write a paragraph explaining their answers or reporting any specific observations they had during the process. We obtained IRB approval and subject consent for sharing findings.

Data analysis

A quantitative analysis was performed to analyze the time that subjects took to perform the assignments, their perceived levels of difficulty in prompting ChatGPT to perform the assignment, and the grades they received on the assignment. The correlation between the time taken and the perceived level of difficulty was calculated using the Pearson correlation coefficient. We also compared the subjects' grades to the grades obtained by the students enrolled in the course (n=54). It is worth noting that students enrolled in the course were asked to complete laboratory assignments in class, but they were given one additional day to submit their work. Students typically completed the homework outside of class. Therefore, we cannot certify that regular students in the course did not use AI tools to generate code for the assignments. Because of the difference of sample size between the two groups (n=3 for subjects explicitely using ChatGPT vs n=54 for students enrolled in the course), no statistical comparison was performed.

A qualitative analysis was also performed to analyze the conversation between the subjects and ChatGPT and their written reflections on the assignment. In particular, we determined the number and type of follow-up prompts that the subjects entered to request changes in the AI-generated code. We define prompt as a text-based input that instructs ChatGPT to generate a response. A thematic analysis was performed by the author to determine the different types of prompts they used to request improvements in their codes using an inductive coding method.

a)



Air Quality Indices

Environmental Engineering

Background Knowledge

$PM_{2.5}$ refers to atmospheric particles (PM) that have a diameter of less than 2.5 micrometers. These particles can come from various sources like animal operations, power plants, motor vehicles, airplanes, residential wood burning, forest fires, volcanic eruptions and dust storms. Since they are so small and light, $PM_{2.5}$ tend to stay longer in the air than heavier particles, increasing the chance of humans inhaling them.

This Photo by Unknown Author is licensed under CC BY

Exposure to fine particles can lead to premature death from heart and lung disease and trigger or worsen chronic disease such as asthma, heart attack, bronchitis and other respiratory problems.
The U.S. Environmental Protection Agency (EPA) regulates air pollution emission and defines the nation's air quality standards. EPA has created an air quality index based on the daily $PM_{2.5}$ concentration, that tells how polluted the air is, along with associated health effects (table1).

Table 1. U.S. Environmental Protection Agency air quality standards for $PM_{2.5}$ pollution

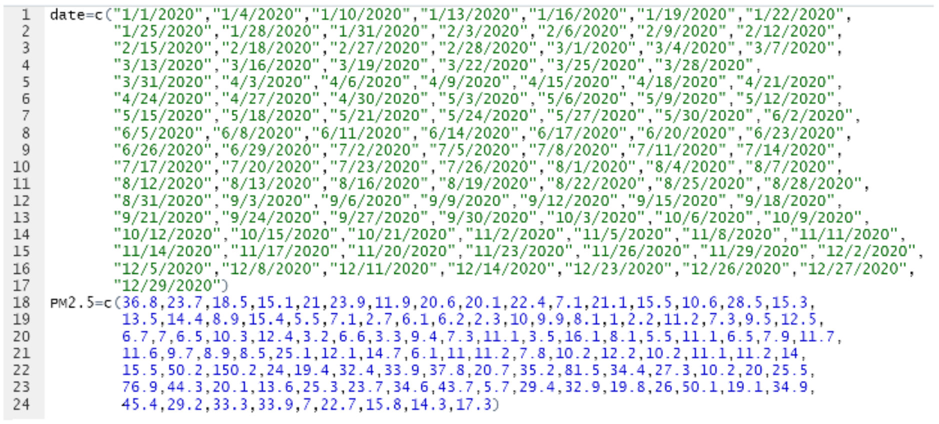
daily $PM_{2.5}$ concentration(µg/m³)	$C < 12.0$	$12.0 \leq C < 35.4$	$35.4 \leq C < 55.4$	$55.4 \leq C < 150.4$	$150.4 \leq C < 250.4$	$250.4 \leq C$
Air Quality Index	Good	Moderate	Unhealthy for Sensitive Groups	Unhealthy	Very Unhealthy	Hazardous

Bakersfield (CA) is the US city with the worst air quality. You have been hired as an intern with EPA to analyze air quality data based on $PM_{2.5}$ concentrations measured in 2020 in Bakersfield.
The data is provided in the file air_quality.R (download here). It contains a vector of dates and a vector of corresponding PM2.5 concentrations in (µg/m³).

Your task is to write a R script that:

- **produces a table with dates, PM2.5 concentrations and corresponding Air Quality Index for each day**
- **Produces a second table with the number of days falling in each air quality index category**

b)



```r
date=c("1/1/2020","1/4/2020","1/10/2020","1/13/2020","1/16/2020","1/19/2020","1/22/2020",
        "1/25/2020","1/28/2020","1/31/2020","2/3/2020","2/6/2020","2/9/2020","2/12/2020",
        "2/15/2020","2/18/2020","2/27/2020","2/28/2020","3/1/2020","3/4/2020","3/7/2020",
        "3/13/2020","3/16/2020","3/19/2020","3/22/2020","3/25/2020","3/28/2020",
        "3/31/2020","4/3/2020","4/6/2020","4/9/2020","4/15/2020","4/18/2020","4/21/2020",
        "4/24/2020","4/27/2020","4/30/2020","5/3/2020","5/6/2020","5/9/2020","5/12/2020",
        "5/15/2020","5/18/2020","5/21/2020","5/24/2020","5/27/2020","5/30/2020","6/2/2020",
        "6/5/2020","6/8/2020","6/11/2020","6/14/2020","6/17/2020","6/20/2020","6/23/2020",
        "6/26/2020","6/29/2020","7/2/2020","7/5/2020","7/8/2020","7/11/2020","7/14/2020",
        "7/17/2020","7/20/2020","7/23/2020","7/26/2020","8/1/2020","8/4/2020","8/7/2020",
        "8/12/2020","8/13/2020","8/16/2020","8/19/2020","8/22/2020","8/25/2020","8/28/2020",
        "8/31/2020","9/3/2020","9/6/2020","9/9/2020","9/12/2020","9/15/2020","9/18/2020",
        "9/21/2020","9/24/2020","9/27/2020","9/30/2020","10/3/2020","10/6/2020","10/9/2020",
        "10/12/2020","10/15/2020","10/21/2020","11/2/2020","11/5/2020","11/8/2020","11/11/2020",
        "11/14/2020","11/17/2020","11/20/2020","11/23/2020","11/26/2020","11/29/2020","12/2/2020",
        "12/5/2020","12/8/2020","12/11/2020","12/14/2020","12/23/2020","12/26/2020","12/27/2020",
        "12/29/2020")
PM2.5=c(36.8,23.7,18.5,15.1,21,23.9,11.9,20.6,20.1,22.4,7.1,21.1,15.5,10.6,28.5,15.3,
         13.5,14.4,8.9,15.4,5.5,7.1,2.7,6.1,6.2,2.3,10,9.9,8.1,1,2.2,11.2,7.3,9.5,12.5,
         6.7,7,6.5,10.3,12.4,3.2,6.6,3.3,9.4,7.3,11.1,3.5,16.1,8.1,5.5,11.1,6.5,7.9,11.7,
         11.6,9.7,8.9,8.5,25.1,12.1,14.7,6.1,11,11.2,7.8,10.2,12.2,10.2,11.1,11.2,14,
         15.5,50.2,150.2,24,19.4,32.4,33.9,37.8,20.7,35.2,81.5,34.4,27.3,10.2,20,25.5,
         76.9,44.3,20.1,13.6,25.3,23.7,34.6,43.7,5.7,29.4,32.9,19.8,26,50.1,19.1,34.9,
         45.4,29.2,33.3,33.9,7,22.7,15.8,14.3,17.3)
```

Figure 1. Example of assignment and R script with provided inputs

Results

Version of ChatGPT used

Subjects 1 and 3 used chatgpt.com while Subject 2 used chatgptfree.ai, except for M12 -3 assignment for which they used chatgpt.com. Subject 2 reported limitations for the chatgptfree.ai site. The chatbot could not follow up on their initial prompts. If they wanted to refine the script, they had to start all over and refine the initial prompt with more information.

Time Taken per Assignment and Perceived Difficulty

The time spent on the programming assignments varied from 8 minutes to 162 minutes across all students and assignments (Appendix A). The subjects reported that they had very little experience with ChatGPT when starting this study and that the first assignment took them a bit longer, with times ranging from 30 to 110 minutes depending on the subject. Subject 1 was the quickest at performing assignments with an average of 16 minutes per assignment, while subject 2 was the slowest with an average of 94 minutes per assignment. Several factors explain this difference. As mentioned earlier, subject 2 was using chatgptfree.ai, which did not allow them to follow up on initial prompts. Every time, they had to rewrite the entire set of instructions and complement them with expected specifications for the code. Another explanation is that subject 2 also ran every version of code generated by ChatGPT in R before deciding whether it needed to be refined or not.

The subjects' perceived level of difficulty of prompting ChatGPT to obtain a satisfactory code ranged from 1/5 to 4/5 depending on the assignment (Appendix A). The average perceived difficulty levels were 1.7/5, 2.5/5 and 1.2/5 for subjects 1, 2, 3, respectively. The highest average degree of difficulty perceived by subject 2 reflects the challenges they experienced with the chatgptfree.ai site. The Pearson correlation coefficients between the time and perceived difficulty were 0.33, 0.50, 0.48, for subjects 1, 2, 3, respectively, indicating low to moderate correlations.

Grades

All three subjects received excellent grades on their assignments (Appendix A). The average of the grades received by each subject ranged from 96 to 98 %. In comparison, the average class grade was 93%. The number of perfect assignments (100%) based on the grading criteria were 6, 6 and 9 for subjects 1, 2 and 3, respectively. Codes that did not receive a 100% score had minor presentation errors such as failure to clear the memory, failure to create variables for all program inputs, and hardcoding some of the inputs. All the codes submitted returned correct calculations or processed data correctly.

The subjects using ChatGPT performed slightly better than the students in the course on all assignments except M8-2 (91.1% for the subjects using ChatGPT and 94.7% for students in the course). For this assignment, Subjects 1 and 2 scores were below the average grade of students enrolled in the course.

Prompting Strategies

The number of prompts to obtain a satisfactory code varied from 1 to 30 across all assignments (Figure 2). There is a lot of variability between subjects. Subjects 1 and 2 prompted ChatGPT 4 times on average while subject 3 prompted 10 times on average.

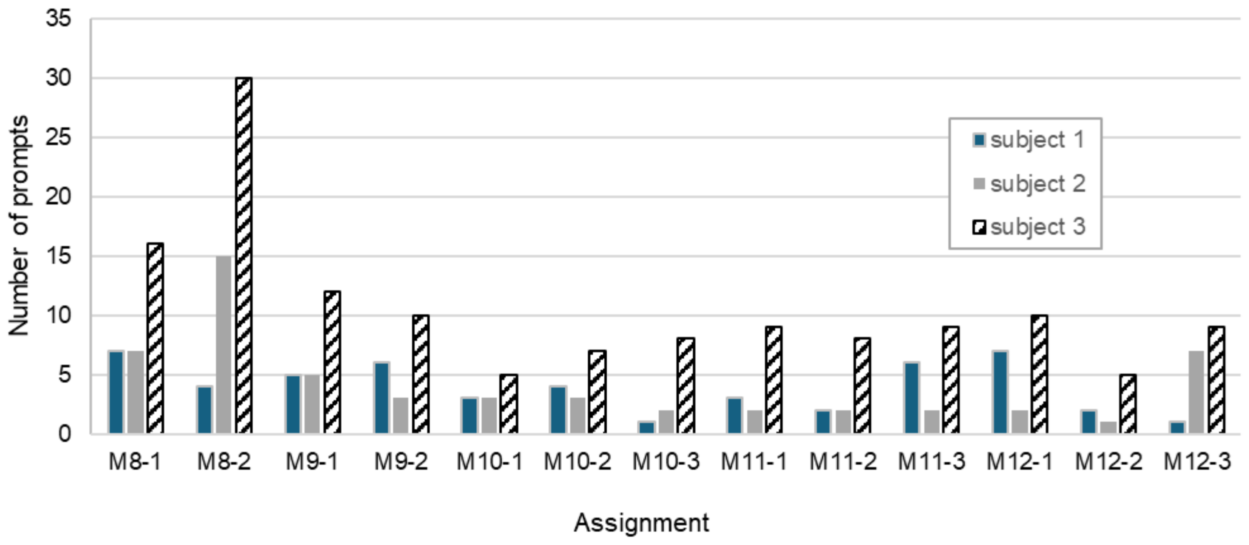


Figure 2. Number of prompts per assignment and user

The prompting strategy also varied greatly between subjects. In their initial prompts, subjects 1 and 3 only provided the problem statement to ChatGPT. However, subject 2 provided additional information that was available to the students enrolled in the course. For module 8 assignments, they provided the assignment grading rubric and asked that the generated code followed the criteria. For the other modules, they provided ten items from the assignment checklist, a document listing good programming practices that students need to emulate in their assignments, which includes elements such as: clearing the memory at the beginning of the script; creating a variable for each problem input; adding comments to explain what each line or chunk of code is doing; reporting answers with correct significant digits and correct units. Subjects' strategies to deal with the provided R scripts containing the input data differed. Subjects 1 and 2 copied and pasted the content of the scripts directly into the chat box, while Subject 3 uploaded the .R scripts to the chat box. This had important implications on how ChatGPT treated this input data.

When the content was pasted directly into the chat box, ChatGPT placed this content directly at the start of the generated code. When the script was uploaded, ChatGPT added a command in the code to source (read) the R script provided, which the student needed to fix later, requiring more prompts.

When writing follow-up prompts to refine the code generated by ChatGPT, subject 2 provided multiple instructions to address multiple issues seen in the previous ChatGPT generated code in one prompt, while subjects 1 and 3 generally wrote one prompt per issue to be resolved.

Types of prompts

The nature of the prompts provided by the subjects beyond the problem descriptions was classified in three types using inductive coding (Figure 3). Because prompts sometimes contained multiple instructions, we counted and assigned a type to each instruction. The three categories were: presentation (improving the presentation of the code), simplification (using simpler commands in the code), and fixing programming errors (that would prevent the code from running or reporting correct answers).

Most of the prompts provided by the subjects focused on improving the script presentation to emulate the best programming practices discussed in the course. This type of prompt corresponded to 40%, 94% and 57% of the prompts of subjects 1, 2 and 3, respectively. (Figure 3). Examples of prompts regularly used by the subjects included: adding code to clear the memory, refining programming comments, or reporting results with appropriate significant figures. As mentioned before, Subject 2 provided ten criteria from the course good programming checklist in their initial prompts, explaining why they primarily used this type of prompt.

The second most used type of prompt dealt with simplifying the code to reflect the programming abilities of a regular student in the course. It corresponded to 50%, 3%, and 40% of the prompts of subjects 1, 2 and 3, respectively. While the initial code provided by ChatGPT performed the correct calculations or tasks in most cases, subjects 1 and 3 opted to ask ChatGPT to update it to make it simpler. Specifically, they asked ChatGPT to avoid using commands or programming structures that had not been covered in the course and to use functions or structures covered in the corresponding module instead. One subject called it “dumbing the code”. For example, they asked to simplify how the results were printed to the console because ChatGPT created sophisticated code using conditional statements to report findings for the first assignment M8-1 (Figure 4). For M11-1 assignment, it was not necessary to write a loop to perform the calculations, but since the topic was loops, subjects 1 and 3 prompted ChatGPT to update the code to include a for loop. The prompting strategy of subjects 1 and 3 to simplify the code evolved during the course of the study. In the first assignments, they prompted ChatGPT to “write simpler code”, or to avoid using a specific command that was not covered in the course.

As their experience with prompting ChatGPT to write or update code increased, they provided more specific prompts such as “use c() to create the vector instead of numeric()”. For assignment M11-3, one subject commented that “ChatGPT kept using commands that I told it not to such as for loops and numeric() for defining an object. I could have done this one quicker myself and I feel that the final product is sort of awkward for this problem.”

The last type of prompt that was used by the subjects focused on asking ChatGPT to fix syntax or programming errors that would have prevented the code from running properly or returning correct inputs. This type of prompt corresponded to 10%, 3% and 3% of the prompts of subjects 1, 2 and 3, respectively. Particularly, subject 2 reported that for assignments M8-2 and M9-2, ChatGPT omitted multiplication signs in the code because they were not written in the formula provided in the problem description, and that they had to prompt ChatGPT to add the multiplication signs. For M12-1 assignment, Subject 1 found that ChatGPT did not correctly interpret the assignment description and wrote conditional statements that failed to consider all the cases listed and returned wrong solutions for four out of fifteen tests (Appendix B). Subject 1 had to write three feedback prompts asking ChatGPT to update the logic before it returned a correct code. For assignments M10-3 and M12-3 that were provided with a starting R script containing inputs, Subject 3 reported that if they uploaded the script and asked ChatGPT to copy the content in the final code generated, it would only copy a portion of the dataset, which would yield false outputs.

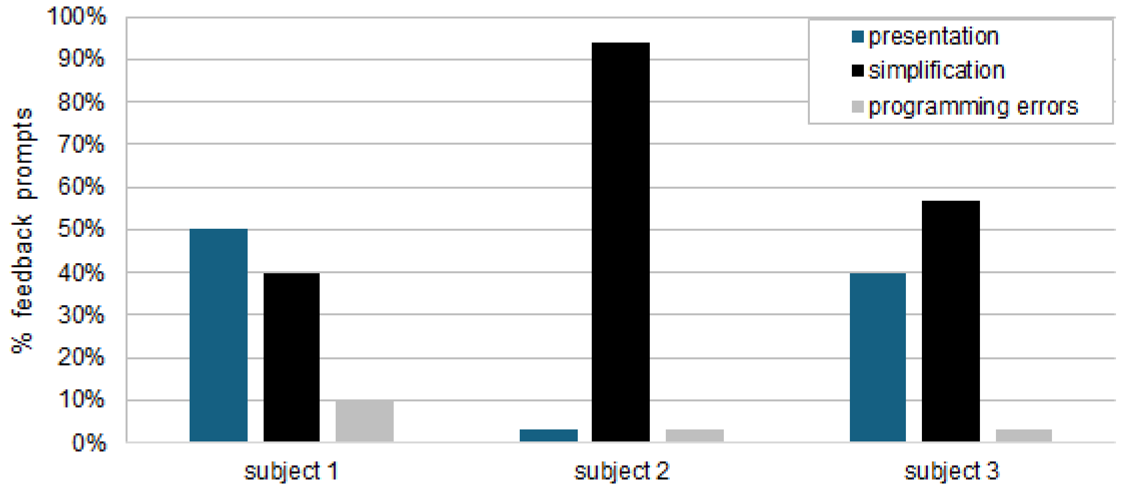
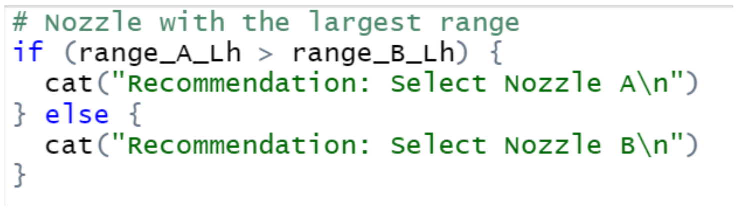


Figure 3. Repartition of prompts per type and subject



```
# Nozzle with the largest range
if (range_A_Lh > range_B_Lh) {
  cat("Recommendation: Select Nozzle A\n")
} else {
  cat("Recommendation: Select Nozzle B\n")
}
```

Figure 4. Example of script created by ChatGPT for the M8-1, first R assignment.

Discussion

An unexpected finding of this study is that the participants put significant effort into prompting ChatGPT to modify the code, ensuring it used only commands covered in class and that it did not use commands that they judged too advanced. Their goal was to produce code that closely resembled what a student in the course would write and to prevent their ChatGPT generated code from being flagged as non-original work. We found no other studies reporting similar behavior among students in programming courses. One possible explanation is that prior research on student use of ChatGPT in programming courses relied on autograders [12] or that human graders focused more on the correctness of outputs rather than on the programming approach. On a positive note, this behavior suggests that participants took the time to analyze the ChatGPT-generated code and requested modifications for the commands they were unfamiliar with, demonstrating critical thinking skills.

We observed distinct prompting strategies among the research subjects. Subject 1 provided extensive information in their initial prompt, including the problem description, grading criteria, and good programming practices that the code should follow. In contrast, Subjects 2 and 3 included only the problem description in their initial prompts and relied on multiple follow-up prompts to address issues in the code one by one. Scholl et al. [13] analyzed the prompting strategies of 200 students using ChatGPT in an introductory programming course and identified similar distinctions. Specifically, they classified three main types of prompting strategies. The first type involves crafting a detailed initial prompt that includes the problem description and explicitly requests a direct solution, resulting in a limited number of follow-up prompts, similar to what subject 1 did. The second type also seeks a direct solution but involves more follow-up prompts to identify and correct mistakes, suggesting that students were not fully satisfied with the initial response, similar to what subjects 2 and 3 did. The third type consists of a larger number of prompts aimed at understanding the AI-generated response and debugging errors, indicating that students used the generative AI tool as a learning partner.

In this study, ChatGPT provided correct code on the first attempt 77% of the time, which is consistent with previous research that tested generative AI’s ability to produce code for programming courses, where the frequency of correct solutions ranged from 52% to 75% [8], [9]. The research subjects were able to detect errors and prompt ChatGPT to fix them, ultimately obtaining correct answers for all assignments. This is also consistent with prior studies that have also observed improvements in ChatGPT-generated code after prompting [8], [9], [11]. Our findings reinforce conclusions from other studies, indicating that ChatGPT and similar generative AI tools do not always produce accurate and relevant responses to coding problems. The students in this study reached the same conclusion. Several studies report that students enrolled in programming courses are also aware of this limitation. For example, 40% of undergraduate students surveyed by Popovici [11] believed generative AI was of little to no help in solving their programming assignments. Similarly, 56% of CS1 students surveyed by Xue et al. [14] held neutral views regarding ChatGPT's ability to provide high-quality solutions for programming assignments. Scholl and Kiesler [15] reported that 31% of the CS1 students that they surveyed rated ChatGPT's accuracy and relevance negatively, while 52% had neutral opinions. Moreover, our findings highlight the importance of teaching our students key skills related to code tracing, comprehension, and testing to assess and evaluate AI-generated code. Educators are increasingly advocating for modifications to introductory programming courses to emphasize these skills [9], [16], [17].

Currently, detecting AI-generated code using publicly available tools is difficult and unreliable, especially for simple code, such as that produced in introductory programming courses [18], [19]. As a result, policies banning AI tools in programming courses cannot be effectively enforced. Instead, instructors should design activities that leverage generative AI to enhance student learning while mitigating its negative impacts. Based on our findings, we identified three course modifications that can be easily implemented in our introductory course. First, students should receive adequate training on using generative AI for coding, including understanding its strengths and limitations and identifying effective prompting strategies. Second, students should still be required to complete several assignments per week without AI and under the supervision of the instructor. This can be enforced during laboratory sessions by setting strict assignment submission deadlines at the end of each session. Lastly, for assignments completed outside of class, grading criteria should explicitly require that students use specific commands covered in the course. This ensures that if students choose to use generative AI, they must still comprehend the code and craft explicit prompts to adjust AI-generated outputs appropriately.

The main limitations of this study include the small number of assignments tested and the limited number of research subjects. Future research could benefit from collecting observations from a larger group of students using ChatGPT for programming assignments to analyze their prompting strategies more comprehensively. Additionally, if we can ensure that the students enrolled in the course complete some assignments without generative AI, such as in supervised laboratory sessions, we could compare assignment performance between students who use generative AI and those who do not. This comparison would provide deeper insights into the impact of AI-assisted coding on learning outcomes.

Conclusions

For the first time, we evaluated ChatGPT's ability to generate code for programming assignments in an authentic introductory programming course for Biological Engineering undergraduates. ChatGPT was prompted by undergraduate students who took the course during the previous year and who had no prior experience using generative AI for writing programs. In this study, ChatGPT provided correct code on the first attempt 77% of the time and achieved 100% accuracy after follow-up prompts. The average grade obtained by the subjects who used ChatGPT were slightly better than the average grade of students enrolled in the course. An unexpected finding was that the subjects asked ChatGPT to modify the code by replacing commands not covered in class with simpler approaches, making the output more similar to that of a student in the course.

The findings from this study offer insights into strategies for integrating generative AI into introductory programming courses to enhance students' programming skills, including code comprehension and testing. Additionally, they suggest ways to minimize opportunities for students to use generative AI to directly obtain complete solutions to their coding assignments.

References

[1] A. R. Ellis and E. Slade, "A new era of learning: considerations for ChatGPT as a tool to enhance statistics and data science education," *Journal of Statistics and Data Science Education*, vol. 31, no. 2, pp. 128–133, 2023.

[2] S. Bubeck *et al.*, "Sparks of artificial general intelligence: Early experiments with GPT-4. arXiv," *arXiv preprint arXiv:2303.12712*, 2023.

[3] N. Cooper, A. T. Clark, N. Lecomte, H. Qiao, and A. M. Ellison, "Harnessing large language models for coding, teaching and inclusion to empower research in ecology and evolution," *Methods in Ecology and Evolution*, 2024.

[4] R. Yilmaz and F. G. Karaoglan Yilmaz, "Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning," *Computers in Human Behavior: Artificial Humans*, vol. 1, no. 2, p. 100005, Aug. 2023, doi: 10.1016/j.chbah.2023.100005.

[5] C. A. G. Da Silva, F. N. Ramos, R. V. de Moraes, and E. L. dos Santos, "ChatGPT: Challenges and benefits in software programming for higher education," *Sustainability*, vol. 16, no. 3, p. 1245, 2024.

[6] E. L. Ouh, B. K. S. Gan, K. Jin Shim, and S. Wlodkowski, "ChatGPT, Can You Generate Solutions for my Coding Exercises? An Evaluation on its Effectiveness in an undergraduate Java Programming Course.," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 2023, pp. 54–60.

[7] R. Budhiraja, I. Joshi, J. S. Challa, H. D. Akolekar, and D. Kumar, "'It's not like Jarvis, but it's pretty close!' - Examining ChatGPT's Usage among Undergraduate Students in Computer Science," in *Proceedings of the 26th Australasian Computing Education*

Conference, in ACE ’24. New York, NY, USA: Association for Computing Machinery, Jan. 2024, pp. 124–133. doi: 10.1145/3636243.3636257.

[8] S. R. Piccolo, P. Denny, A. Luxton-Reilly, S. Payne, and P. G. Ridge, “Many bioinformatics programming tasks can be automated with ChatGPT,” *arXiv preprint arXiv:2303.13528*, 2023.

[9] C. Geng, Y. Zhang, B. Pientka, and X. Si, “Can chatgpt pass an introductory level functional language programming course?,” *arXiv preprint arXiv:2305.02230*, 2023.

[10] J. Savelka, A. Agarwal, C. Bogart, Y. Song, and M. Sakr, “Can generative pre-trained transformers (gpt) pass assessments in higher education programming courses?,” in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 2023, pp. 117–123.

[11] D.-M. Popovici, “ChatGPT in the classroom. Exploring its potential and limitations in a Functional Programming course,” *International Journal of Human–Computer Interaction*, vol. 40, no. 22, pp. 7743–7754, Nov. 2024, doi: 10.1080/10447318.2023.2269006.

[12] C. L. Gordon, R. Lysecky, and F. Vahid, “The Rise of Program Auto-grading in Introductory CS Courses: A Case Study of zyLabs,” presented at the 2021 ASEE Virtual Annual Conference Content Access, Jul. 2021. Accessed: Jan. 13, 2025. [Online]. Available: https://peer.asee.org/the-rise-of-program-auto-grading-in-introductory-cs-courses-a-case-study-of-zylabs

[13] A. Scholl, D. Schiffner, and N. Kiesler, “Analyzing Chat Protocols of Novice Programmers Solving Introductory Programming Tasks with ChatGPT,” May 29, 2024, *arXiv*: arXiv:2405.19132. doi: 10.48550/arXiv.2405.19132.

[14] Y. Xue, H. Chen, G. R. Bai, R. Tairas, and Y. Huang, “Does ChatGPT Help With Introductory Programming?An Experiment of Students Using ChatGPT in CS1,” in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, in ICSE-SEET ’24. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 331–341. doi: 10.1145/3639474.3640076.

[15] A. Scholl and N. Kiesler, “How Novice Programmers Use and Experience ChatGPT when Solving Programming Exercises in an Introductory Course,” Jul. 30, 2024, *arXiv*: arXiv:2407.20792. doi: 10.48550/arXiv.2407.20792.

[16] S. Lau and P. Guo, “From ‘Ban It Till We Understand It’ to ‘Resistance is Futile’: How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot,” in *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, in ICER ’23, vol. 1. New York, NY, USA: Association for Computing Machinery, Sep. 2023, pp. 106–121. doi: 10.1145/3568813.3600138.

[17] C. Zastudil, M. Rogalska, C. Kapp, J. Vaughn, and S. MacNeil, “Generative AI in Computing Education: Perspectives of Students and Instructors,” in *2023 IEEE Frontiers in Education Conference (FIE)*, Oct. 2023, pp. 1–9. doi: 10.1109/FIE58773.2023.10343467.

[18] J. Wang, S. Liu, X. Xie, and Y. Li, “Evaluating AIGC Detectors on Code Content,” Apr. 11, 2023, *arXiv*: arXiv:2304.05193. doi: 10.48550/arXiv.2304.05193.

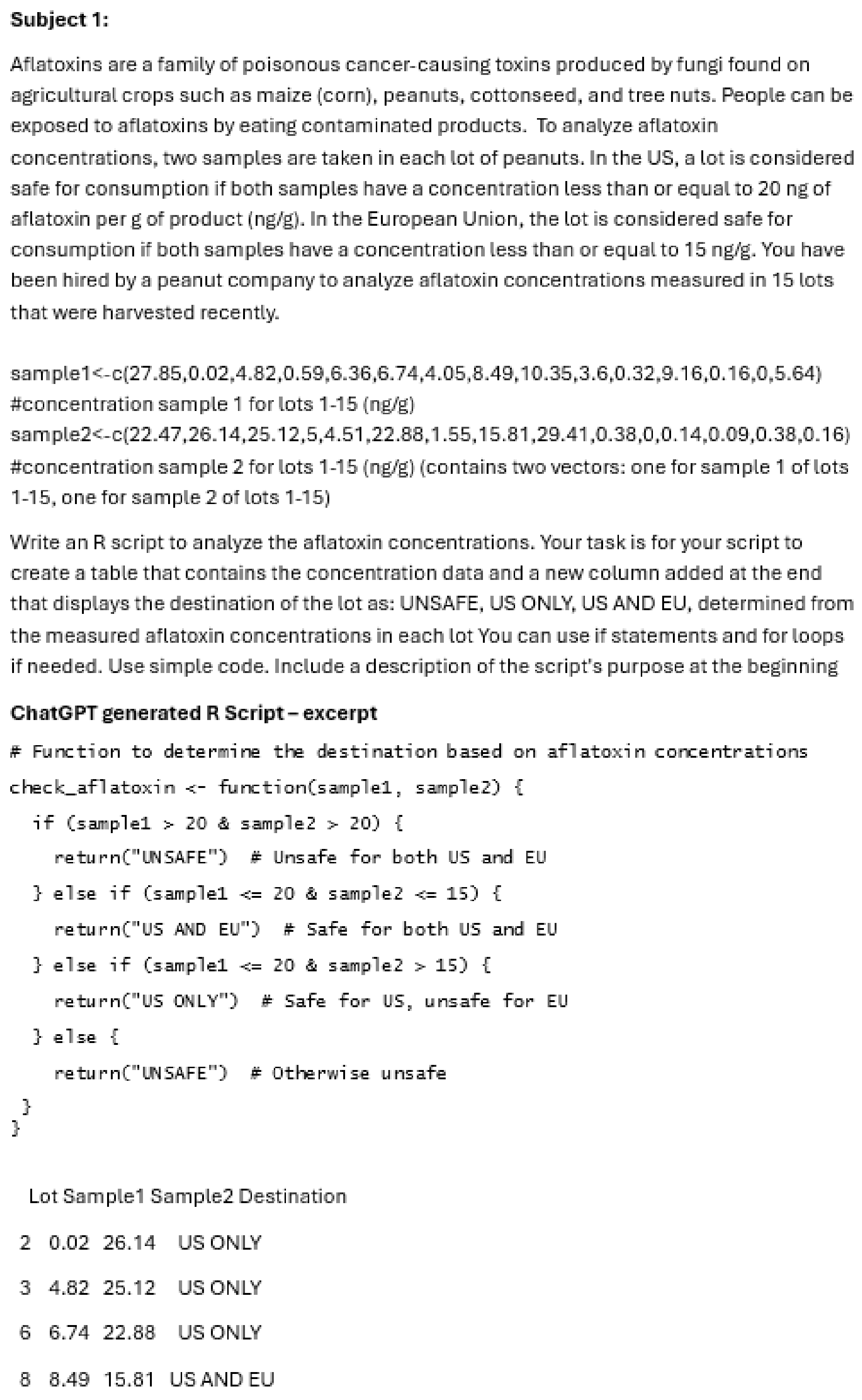
[19] M. Hoq *et al.*, “Detecting ChatGPT-Generated Code Submissions in a CS1 Course Using Machine Learning Models,” in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, in SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, Mar. 2024, pp. 526–532. doi: 10.1145/3626252.3630826.

Appendix A. Time taken, perceived difficulty of prompting ChatGPT, grade obtained

	subject 1			subject 2			subject 3			Subjects average	Class average
	Time taken (min)	Perceived difficulty (/5)	Grade (%)	Time taken (min)	Perceived difficulty (/5)	Grade (%)	Time taken (min)	Perceived difficulty (/5)	Grade (%)	Grade (%)	Grade (%)
M8 1	30	1	95.6	110	3	100.0	45	1	100.0	98.5	95.7
M8 2	15	1	90.0	142	3	87.8	60	2	95.6	91.1	94.7
M9 1	20	2	100.0	162	3	100.0	66	1	100.0	100	98.8
M9 2	20	1	85.6	76	2	88.9	19	1	86.7	87	81.3
M10 1	15	1	95.6	62	2	93.3	50	1	95.6	94.8	94.8
M10 2	20	3	100.0	94	3	97.8	25	1	100.0	99.3	96.4
M10 3	10	1	100.0	124	3	100.0	47	2	100.0	100	91.2
M11 1	10	1	100.0	57	3	97.8	23	1	100.0	99.3	92.8
M11 2	10	1	100.0	76	2	100.0	24	1	100.0	100	92
M11 3	20	4	97.8	120	3	100.0	28	1	100.0	99.3	88.1

M12 1	20	3	93.3	66	3	100.0	25	1	100.0	97.8	94.5
M12 2	8	1	95.6	59	2	97.8	13	1	95.6	96.3	94.6
M12 3	10	2	100.0	79	1	95.6	43	2	100.0	98.5	95.3

Appendix B. Initial prompt provided by the student and excerpt of the code provided by ChatGPT for assignment M12-1. The table at the bottom represents outputs from the code that are not correct (lot 2, 3 and 6 correct destination is UNSAFE, and lot 8 correct destination is US only).



Subject 1:

Aflatoxins are a family of poisonous cancer-causing toxins produced by fungi found on agricultural crops such as maize (corn), peanuts, cottonseed, and tree nuts. People can be exposed to aflatoxins by eating contaminated products. To analyze aflatoxin concentrations, two samples are taken in each lot of peanuts. In the US, a lot is considered safe for consumption if both samples have a concentration less than or equal to 20 ng of aflatoxin per g of product (ng/g). In the European Union, the lot is considered safe for consumption if both samples have a concentration less than or equal to 15 ng/g. You have been hired by a peanut company to analyze aflatoxin concentrations measured in 15 lots that were harvested recently.

sample1<-c(27.85,0.02,4.82,0.59,6.36,6.74,4.05,8.49,10.35,3.6,0.32,9.16,0.16,0,5.64)
#concentration sample 1 for lots 1-15 (ng/g)
sample2<-c(22.47,26.14,25.12,5,4.51,22.88,1.55,15.81,29.41,0.38,0,0.14,0.09,0.38,0.16)
#concentration sample 2 for lots 1-15 (ng/g) (contains two vectors: one for sample 1 of lots 1-15, one for sample 2 of lots 1-15)

Write an R script to analyze the aflatoxin concentrations. Your task is for your script to create a table that contains the concentration data and a new column added at the end that displays the destination of the lot as: UNSAFE, US ONLY, US AND EU, determined from the measured aflatoxin concentrations in each lot You can use if statements and for loops if needed. Use simple code. Include a description of the script's purpose at the beginning

ChatGPT generated R Script – excerpt

```
# Function to determine the destination based on aflatoxin concentrations
check_aflatoxin <- function(sample1, sample2) {
  if (sample1 > 20 & sample2 > 20) {
    return("UNSAFE")  # Unsafe for both US and EU
  } else if (sample1 <= 20 & sample2 <= 15) {
    return("US AND EU")  # Safe for both US and EU
  } else if (sample1 <= 20 & sample2 > 15) {
    return("US ONLY")  # Safe for US, unsafe for EU
  } else {
    return("UNSAFE")  # Otherwise unsafe
 }
}
```

```
  Lot Sample1 Sample2 Destination
 2   0.02  26.14    US ONLY
 3   4.82  25.12    US ONLY
 6   6.74  22.88    US ONLY
 8   8.49  15.81   US AND EU
```