# Prompt Problems: A New Programming Exercise for the Generative AI Era


Paul Denny
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Juho Leinonen
University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.nz

James Prather
Abilene Christian University
Abilene, TX, USA
james.prather@acu.edu

Andrew Luxton-Reilly
University of Auckland
Auckland, New Zealand
a.luxton-reilly@auckland.ac.nz

Thezyrie Amarouche
University of Toronto Scarborough
Toronto, ON, Canada
thezyrie.amarouche@mail.utoronto.ca

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Brent N. Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu



## ABSTRACT

Large language models (LLMs) are revolutionizing the field of computing education with their powerful code-generating capabilities. Traditional pedagogical practices have focused on code *writing* tasks, but there is now a shift in importance towards reading, comprehending and evaluating LLM-generated code. Alongside this shift, an important new skill is emerging – the ability to solve programming tasks by constructing good prompts for code-generating models. In this work we introduce a new type of programming exercise to hone this nascent skill: 'Prompt Problems'. Prompt Problems are designed to help students learn how to write effective prompts for AI code generators. A student solves a Prompt Problem by crafting a natural language prompt which, when provided as input to an LLM, outputs code that successfully solves a specified programming task. We also present a new web-based tool called Promptly which hosts a repository of Prompt Problems and supports the automated evaluation of prompt-generated code. We deploy Promptly in one CS1 and one CS2 course and describe our experiences, which include student perceptions of this new type of activity and their interactions with the tool. We find that students are enthusiastic about Prompt Problems, and appreciate how the problems engage their computational thinking skills and expose them to new programming constructs. We discuss ideas for the future development of new variations of Prompt Problems, and the need to carefully study their integration into classroom practice.



## CCS CONCEPTS

• **Social and professional topics → Computing education**; **CS1**.





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



ACM ISBN 979-8-4007-0423-9/24/03.
https://doi.org/10.1145/3626252.3630909

## KEYWORDS

AI code generation; artificial intelligence; generative AI; large language models; LLMs; prompt engineering; prompt problems

**ACM Reference Format:**
Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Prompt Problems: A New Programming Exercise for the Generative AI Era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630909


## 1 INTRODUCTION

The advent of large language models (LLMs) is having a rapid and significant impact on computing education practice, particularly at the introductory level [25]. Traditional pedagogical approaches have focused on helping students learn how to *write* code. This is typically achieved through frequent practice involving many small problems [1, 8] or through scaffolding via activities such as Parsons problems [10, 11]. However, LLMs are now capable of producing code automatically and have demonstrated impressive performance on problems that are typical in introductory programming courses [13, 14, 27]. Despite the opportunities that LLMs may afford, educators have voiced concerns around potential misuse of these models for plagiarism, as well as over-reliance by beginners on AI-generated code [3], leading to a possible erosion of traditional coding skills [9]. New pedagogical approaches are needed to develop the changing skillsets that students require in the era of generative AI [6].

Teaching students to read and understand code are longstanding goals of introductory courses, and they are becoming increasingly important skills given the ease with which code can be generated by LLM-based tools. An equally important emerging skill is *the ability to formulate effective prompts for LLMs to generate code*. Indeed, coding via natural language may vastly increase end-user programming activities across a wide range of applications and tasks [28]. Recent work has shown that although many typical introductory problems can be solved by LLMs using verbatim textbook or exam problem statements [13, 14], this approach is not always sufficient. For example, manual modification of the prompts to include explicit algorithmic hints greatly improves code-generation performance [30]. In recent work, Denny et al. argue that the ability to engineer effective prompts is now an essential skill for computing students, although they do not propose concrete approaches for how this can be taught [7].

In the current paper we introduce the concept of a 'Prompt Problem' – a new exercise paradigm in which students solve programming exercises by formulating natural language prompts for code-generating LLMs. Students are presented with a representation of a problem that illustrates how input values should be transformed to an output. Their task is to devise a prompt that guides an LLM to generate the code required to solve the problem.

In addition to conceptualizing the problem type, we make two other contributions in this work: (1) we introduce a tool (called Promptly) for delivering Prompt Problems, that displays a problem representation, converts a prompt written by a student to code (via an API call to an LLM), and then executes the code against a suite of test cases; and (2) we present our observations from deploying Prompt Problems to programming students in a CS1 course and a CS2 course, and reflect on our experiences of using them in our teaching for the first time.

## 2 RELATED WORK

Early work studying LLMs in computing education centered on their capabilities, largely driven by concerns that they would lead to a flood of cheating [23] and the effect that would have on student learning. Sometimes, such work involved comparing LLM and student performance, for example in generating explanations of code [18]. Finnie-Ansley et al. demonstrated that Codex (based on GPT-3) ranked in the top quartile of real introductory programming (CS1) students on real exams [13]. A year later Finnie-Ansley et al. extended this work to data structures and algorithms (CS2) exams with very similar results [14]. Other studies on the capabilities of LLMs have revealed impressive proficiency in dealing with object-oriented programming tasks [5], Parsons problems [27], mathematical questions for computer graphics [12], and compiler error messages [19]. Many of these explorations also revealed that LLMs are not infallible and can produce solutions that do not align with best programming practice [5], struggle with longer and higher-level specifications [13], and cause students to become confused when reading code that they did not write themselves [16, 26]. Babe et al. even found that LLMs can mislead students, causing them to believe that their own prompts are more (or less) effective than they are in reality [2].

Recently, the focus has started to shift from assessing the capabilities of LLMs to using them in teaching and learning practice [21]. For example, Sarsa et al. showed that LLMs can generate viable programming exercises including test cases and explanations [29], and Liffiton et al. describe the use of an LLM-powered teaching assistant with guardrails suitable for computing courses [20]. There is growing acceptance for the use of AI in the classroom. Lao and Guo interviewed 19 introductory programming instructors from nine countries across six continents and found that some instructors are embracing the idea of integrating AI tools into current

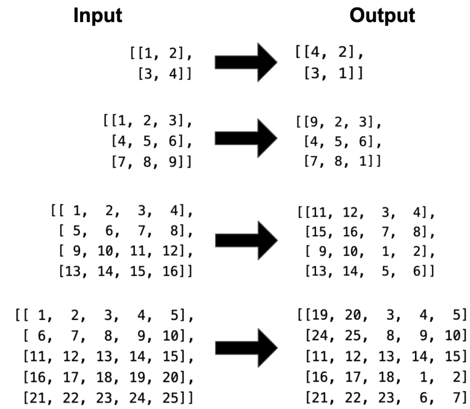


**Figure 1: An example Prompt Problem that displays the data visually to prevent copying and pasting of the description into an LLM. The goal is to swap the top-left and bottom-right non-overlapping quadrants of the matrix.**

courses via mechanisms such as giving personalized help to students and aiding instructors with time-consuming tasks [17]. New resources are also being developed, with one notable example being the recent textbook by Zingaro and Porter for teaching introductory programming using Copilot and ChatGPT [24].

A logical next step towards integrating LLMs into teaching practice is the development of new tools to aid students in effectively working with LLMs for learning. MacNeil et al. used LLM-generated code explanations successfully in a web software development e-book [22], and Jury et al. describe a tool that automatically generates interactive worked examples for students learning programming [15]. Further integration of LLMs into computing courses seems inevitable and stands to transform the way the subject is taught at all levels [6, 31]. We believe that Prompt Problems will be one important step along the journey towards regular use of LLMs in computing education.

## 3 PILOT STUDY

To motivate the need for our work, and to understand how students might use LLM tools like ChatGPT to communicate program requirements, we asked a group of graduate students at the University of Auckland to participate in a prompt writing assignment pilot study. This assignment took place during a single class session in April 2023. We provided a visual representation of a problem (see Fig. 1) and asked participants to query ChatGPT to write a program that could convert the shown inputs to the corresponding example outputs. The problem description was provided visually to prevent participants from easily copying and pasting it and, instead, to encourage them to formulate a suitable prompt themselves.

Fifteen graduate students participated in the pilot, completing the activity described above, reflecting on it by writing an open-response review of the task, and opting to share their work with us. We expected computer science graduate students to have few problems writing effective prompts, however this was not the case. Students wrote incomplete prompts (e.g. *"I have a square matrix, and I want to swap the first half of the rows with the second half of*

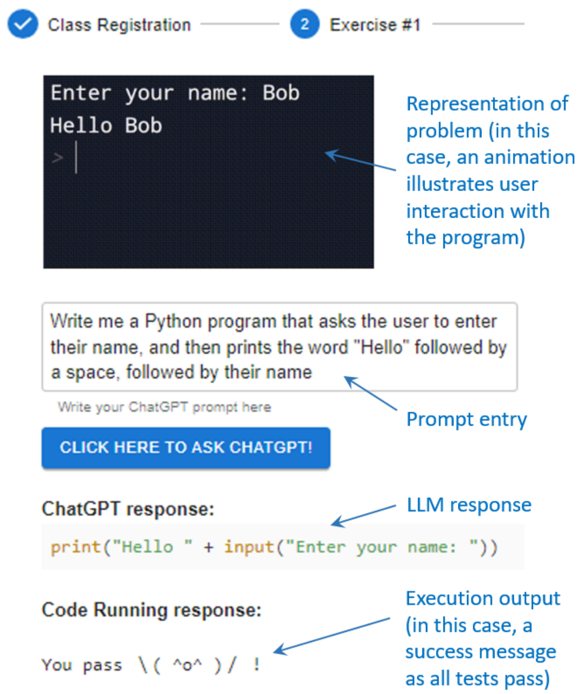


**Figure 2: Interface layout for a Prompt Problem within the web-based Promptly tool (with figure annotations added in blue). The layout is compressed for space reasons.**

*the rows"*), tried to engage in conversations with the tool to refine the generated code, and tried to solve the wrong problem (e.g. *"give me a function which works by first swapping the elements of each row in place, and then swapping the elements of each column in place"*). It became apparent to us that students, even at the graduate level, could benefit from explicit prompt writing practice that could teach them to understand the problem, write a single thorough prompt, and check the code generated by the LLM as having complete test case coverage. We therefore propose the idea of Prompt Problems to address this new gap in programming education.

## 4 PRACTICING PROMPT PROBLEMS

To deliver Prompt Problems to students, we developed a web-based tool called Promptly. In the current implementation, the code generated by the LLM is not editable so the prompt must be complete and self-contained. Other variations are possible and we discuss these in Section 6.1.

### 4.1 Tool Design

Within the Promptly tool, sets of Prompt Problems are organized into course repositories which students select after logging in. Each Prompt Problem within a course repository consists of a visual representation of a problem – an image that does not include a textual description of the problem – and a set of associated test cases that are used to verify the code that is generated by the LLM.

When viewing a Prompt Problem, the student is shown the visual representation of the problem, and a partial prompt to complete. For problems where the solution is a Python program, this partial prompt begins: "Write a Python program that...", to guide the student. If the problem requires students to write a single function, then the partial prompt is: "Write a Python function called...". When any text is entered, the "Click here to ask ChatGPT!" button is enabled, and clicking this button constructs a prompt that is sent to the LLM. This prompt consists of the verbatim text entered by the student, as well as some additional prompting to guide the model to produce only code and no additional explanatory text.

Once the response is received from the LLM, it is then sent to a sandbox for execution against a test suite. We use the publicly available sandbox associated with the CodeRunner tool[1]. If the generated code passes the tests for the Prompt Problem, then the student receives a success message and is directed to progress to the next problem. If any of the test cases fail, then the first failing test case is shown to the student. They are then able to edit the prompt and resubmit in order to generate a new code response.

Figure 2 shows a screenshot of the tool interface (slightly compressed for space reasons). In the screenshot, the learner has logged in, selected their course and exercise, and has entered a prompt that successfully solves the problem. In our implementation, students must solve each problem in order to progress to the next problem.

### 4.2 Classroom Evaluation

Prompt Problems are a novel task for learners in programming courses, and we are interested in understanding what students think about them. *They are also novel for instructors* – and so we are particularly interested in understanding whether the problems we have created are appropriately challenging.

We deployed Promptly as an ungraded (i.e. optional) laboratory task in two Python-based courses (one CS1 and one CS2) taught at the University of Auckland, New Zealand. The CS1 lab was conducted in the second week of the course, at which point students were writing single-file scripts, without the use of functions, and had learned about standard input and output, arithmetic, and conditional statements. For the CS2 course, the lab was also conducted in the second week of the course and all students in this course were familiar with the concept of functions.

Three problems were available on Promptly for each course. Table 1 provides a very brief description of each problem (note, these descriptions were not shown to students but are listed here for the benefit of the reader) alongside one example that illustrates one input with a corresponding output. The CS1 problems all required the generation of a program that processed standard input and printed output, whereas the CS2 problems all required a function that returned a value. The first problem in the CS1 course was the problem previously illustrated in Figure 2. To evaluate the first use of Prompt Problems in our teaching, we explore the following two questions around how students interact with the problems and their opinions on this new type of learning activity:

- When solving Prompt Problems, how many attempts do students require and to what extent do successful prompts vary in terms of length?
- What are students' perceptions of Prompt Problems and on learning programming through constructing prompts for LLMs?

[1] https://github.com/trampgeek/jobe

**Table 1: Summary of student interactions with the Prompt Problems. For each problem, a brief description and example is shown (the description is for the benefit of the reader and was not presented to students). The total number of students (Students) who successfully solved each problem is given (the % shown in parentheses is the percentage of students attempting the problem who successfully solved it). Also shown is the average number of submissions (Sub) these students required, as well as the mean, minimum and maximum number of words used in successful prompts.**

| Problem | Description | Example | Students | Sub | Mean | Min | Max |
|---|---|---|---|---|---|---|---|
| CS1-1 | Display a greeting and the user's name (e.g. see Fig. 2) | Input: Serena → Hello Serena | 44 (76%) | 2.3 | 18.0 | 7 | 33 |
| CS1-2 | Classify an age using a set of four labels | Input: 14 → Teenager | 31 (86%) | 1.8 | 47.9 | 26 | 85 |
| CS1-3 | Average the 3 middle values in a set of 5 values | Input: 8.0 9.5 7.5 6.0 9.0 → 8.17 | 20 (65%) | 7.5 | 40.7 | 25 | 66 |
| CS2-1 | Count the number of occurrences of 0 in a list | counter([0, 2, 3, 4, 0]) → 2 | 136 (75%) | 2.4 | 23.0 | 10 | 84 |
| CS2-2 | Extract the first letter of each word in input string | initials('abc def ghi') → 'ADG' | 121 (96%) | 1.3 | 28.3 | 12 | 88 |
| CS2-3 | Create a list with element occurrences equaling values | repeat([2, 0, 1, 3]) → [2, 2, 1, 3, 3, 3] | 114 (99%) | 1.5 | 34.2 | 16 | 92 |

For the three Prompt Problems in each course we investigate the number of prompt submissions required to solve each one and the number of words used in the submitted prompts. To gauge student perceptions of solving Prompt Problems, students in both courses were invited to provide feedback on their experience. This feedback was not graded, and was given in response to the following prompt: *"We would appreciate hearing about your experiences completing the exercises and in particular, how you think the experience of writing prompts may help you to learn programming"*.

## 5 EXPERIENCES

The courses in which Prompt Problems were used were taught in July 2023, and participation by students was optional. A total of 58 (out of 414 enrolled) students in the CS1 course and 182 (out of 444 enrolled) students in the CS2 course chose to attempt at least one problem on Promptly.

### 5.1 Student Interactions with Prompt Problems

As summarized in Table 1, in the CS1 course participants submitted 2.3 attempts (on average) for Problem 1, 1.8 for Problem 2, and 7.5 for Problem 3. Given that only students who were successful on Problems 1 and 2 progressed to Problem 3, this last problem appeared to be the most difficult. The visual representation of this problem showed a row of five people (stylized as judges of a competition) holding up score cards with the maximum and minimum scores crossed out. Listing 1 shows three prompts that were submitted by different students attempting Problem 3 in the CS1 course (CS1-3). Some students found it difficult to infer the goal from the problem representation. For example, in the first prompt shown in Listing 1 the student has incorrectly inferred that values included in the average calculation should be sufficiently close to their predecessors. The length of this incorrect prompt is 101 words – in comparison the lengths of the *correct* prompts for this problem ranged from 25 to 66 words.

In the second example in Listing 1, the student has not attempted to provide a prompt that demonstrates they have understood what the problem is asking, but instead they have created a prompt that simply parrots back to the tool the three example tests cases shown in the problem description. The student then asks the model: *"Can you please replicate this program?"*. The student submitted this prompt four times in a row, but all attempts were unsuccessful. Finally, the third example in Listing 1 is the shortest successful prompt that was submitted for this problem (25 words).

Overall, the average number of words in successful prompts for the three CS1 problems was 18.0, 47.9, and 40.7. In comparison, average successful prompt lengths for the CS2 problems were 23.0, 28.3 and 34.2. We observed a consistent reduction in the number of students solving subsequent problems in each course – this was not unexpected given the optional nature of the activity. Success rates were particularly high in the CS2 course, with almost all students who progressed to Problems 2 and 3 solving them (with, on average, fewer than two submissions).

Figures 3 and 4 show fine-grained submission patterns for the first problem in each course (CS1-1 and CS2-1, respectively). Similar figures for all other problems are available as an online appendix[2]. Each line on these figures represents the submissions made by one student, illustrating how the word lengths of the prompts changed over time. All successful submissions are highlighted with a blue dot; for students who did not solve the problem, the final unsuccessful submission is shown with an orange X. Most students stopped working on a problem as soon as they solved it, although some continued working and experimenting with different prompts.

In both figures, it is clear that many students solved the problem on their very first attempt (a single blue dot at submission 1). An interesting observation here is the considerable variation in prompt length across these successful submissions. It is likely that some of the longer prompts are not as succinct as they could be, which suggests some students may not be leveraging the power of the LLMs to their full extent. As an example, the shortest successful prompts to CS2-2 and CS2-3 were the 12-word and 16-word prompts: *"I want a function called initials which returns initials of the sentence"* and *"Write me a Python3 function called repeat(list) which repeats the value according to its value"*. In comparison, the longest successful prompts for these problems were 88 and 92 words, respectively. Future variations of this activity could require that students submit working prompts that are less than some target length, to encourage them to be efficient with their word use. Future work may also wish to reward students for the *robustness* of their prompts, by calculating how frequently correct code is generated if the prompt is submitted multiple times.

[2]https://osf.io/cw5gb/?view_only=343aeadc743047beb85764984ca1258b

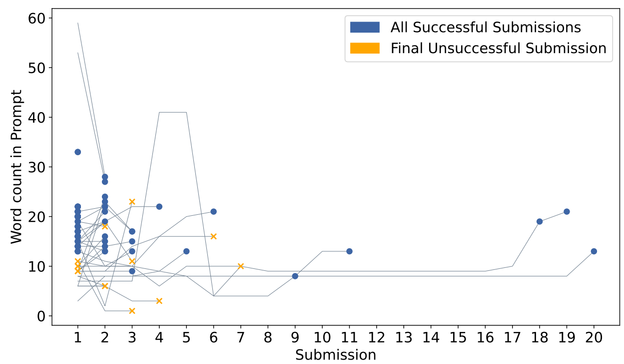


**Figure 3: Each line represents all submissions made by a student for the CS1-1 problem. Blue dots denote every successful submission; an orange X denotes final unsuccessful submission. Several students submit more than one successful prompt, indicating experimentation with the problem.**

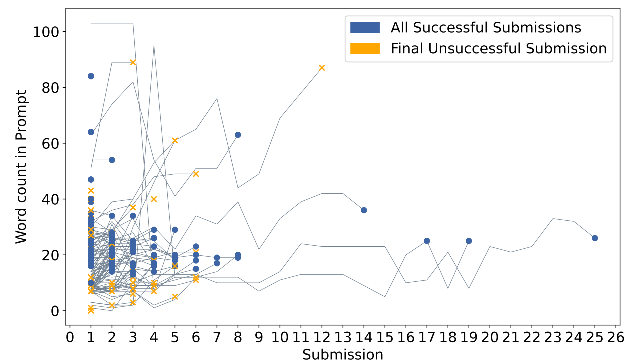


**Figure 4: Each line represents all submissions made by a student for the CS2-1 problem.**

**Listing 1** Three student-submitted prompts for CS1-3

**Misinterpreting the problem:**
Write me a Python program that does the following:
1. Prompts the user to enter five decimal numbers (1dp) between 1.0 and 10.0 separated by spaces.
2. Chooses three of these numbers using the following rules: a number chosen be different from the previously chosen numbers and each subsequently chosen value must be within 0.5 of its predecessor. If the user has not provided numbers that sufficiently meet this criteria, call them an idiot and prompt them for another five values.
3. Find the average of these numbers and round the result to 2dp. Precede this result with the numbers chosen.

**Parroting the tests:**
A Python program requests the user "enter five decimal numbers (separated by spaces)". In the first example the user inputs the five numbers 2.0 3.0 3.0 3.0 4.0 to which the program outputs 3.0. In the second example the user inputs the five numbers 8.0 9.5 7.5 6.0 9.0 to which the program outputs 8.17 . In the third example the user inputs the five numbers 4.0 6.5 8.0 7.0 6.0 to which the program outputs 6.5. Can you please replicate this program?

**Successful:**
Write me a Python program that takes five decimal number separated by spaces, and outputs the average of the 3 median numbers rounded to 2dp.

## 5.2 Student Reflections on Prompt Problems

Of all the students who attempted at least one Prompt Problem in either course, a total of 153 chose to provide a response to the open-ended reflection question. As this activity was new to students in both courses, we analyzed their feedback in combination. We report the main themes that emerged from our analysis below.

*5.2.1 Exposure to new coding constructs.* As our evaluation was conducted early in both courses, the generated code would sometimes contain features that were unfamiliar to students. For the most part, students commented positively on this aspect, and a theme emerged around how these problems would introduce students to new programming constructs and techniques. As one CS1 student commented: *"These exercises introduced me to new functions... so this method of writing code could help increase my programming vocabulary"*. Similar feedback was provided by students in the CS2 course, even though they had prior programming experience: *"[Promptly] could find condensed ways to solve them using Python3's inbuilt functions, some even we have not been taught yet."*

One student commented on the value of seeing both the structure and syntax of the code generated by the LLM: *"The main benefit I gained ... was observing the logical structure of the programs that it created. In all three cases it used functions that I was previously unaware of, allowing me to gain an understanding of how they could be used and the correct syntax for implementing them."*

*5.2.2 Enhancing computational thinking.* Constructing prompts that clearly describe the steps needed to solve a problem draws on computational thinking skills. This was noted in the student reflections, as illustrated by the following quote from a CS2 student: *"I do think that writing prompts for code is a good way of developing analytical and problem-solving thinking and skills as it forces you to think through the steps needed to take the input through to the output"*.

Several participants found that writing prompts helped them improve their problem-solving skills, as they could focus on the logic required rather than low-level syntax: *"I think while writing prompts for AI, we actually have to have a clear logic to break down the question and explain in plain words"* and *"Gaining experience from writing prompts can help me become a more effective programmer by allowing me to generate the necessary code while focusing solely on the logic of the code I want to create"*.

*5.2.3 Resistance and negative feedback.* Although generally positive statements about the activity were more common (e.g. *"That was really fun! I loved the exercise and I feel like it would help me significantly in future labs"*), some students appeared resistant to taking part, citing fears about potential impacts on their creativity. One student expressed: *"I don't have much intention of using ChatGPT at the moment as I major in design and I have a strong belief in personal creativity"*. Another was more blunt: *"I refuse to use chatGPT for programming"*. Over-reliance on AI generated outputs is a commonly cited concern within the education community, and several students commented on this aspect, including: *"it is critical for students to learn the ability to write code independently rather than relying only on AI-generated answers"* and *"I feel like it is too tempting of a tool to use through the labs and not learn and develop these skills yourself"*. These concerns align with previous work that has looked into students' opinions on AI code generation [26].

Further exploring these concerns is an essential avenue for ongoing work, given that some students appeared quite anxious about their future as computing professionals. Upon reflecting on the Prompt Problems task, one student felt that there would no longer be a need for expertise in programming: *"I don't think its a stretch to imagine that in the future 'programmers' won't even be needed and will be replaced by someone who is able to write instructions for the program they want to make. I would be lying if I said I wasn't worried about the future of the majority of programming jobs."* Another student, in the CS2 course, commented on the emotional impact of the task and expressed rather bleak views of the future: *"You have just ruined every piece of self esteem I had regarding coding. I know full well that it would have taken me around 35 minutes to figure out how to create those functions and that damn computer did it in seconds. Robots are going to own us within years."* Overall, while most students reported finding Prompt Problems beneficial, particularly for exposure to new programming constructs and for strengthening computational thinking skills when explaining problems, a minority of students were both hesitant and concerned about the use of generative AI tools for learning programming.

# 6 DISCUSSION

In contrast to other tools students use, such as compilers, learning to use LLMs presents unique challenges. For example, we do not need to worry about teaching students that compilers might sometimes make a mistake, and yet the literature documents the difficulty students have with compiler error messages [4, 19]. In contrast, identical input prompts to an LLM can produce different outputs, and these can sometimes be both syntactically and semantically incorrect. Deliberate exposure to the inconsistencies of LLMs, such as through practice with Prompt Problems, can serve to highlight the importance of a "critical eye" in evaluating generated code and may help to moderate potential over-reliance on these tools.

Although Promptly evaluates prompt effectiveness in producing correct programs, it does not evaluate the efficiency of the prompts. Our unit tests consider only whether the given inputs are translated to the expected outputs. A prompt could include irrelevant words and generate irrelevant code constructs, and as long as it still translates the inputs to the expected outputs, our tool will treat the task as completed successfully. Future work should address how to go beyond effective prompts to efficient (and effective) prompts.

As this was our first experience deploying Prompt Problems to students, participation was optional. Students could also only attempt a problem if they had successfully solved the previous one. Thus, there is likely considerable self-selection bias in our data. Nevertheless, early feedback from students was mostly positive. Future work should aim to expose Prompt Problems to a broader range of students, and provide incentives for their completion.

## 6.1 Variations and Problem Design

There are various ways that Prompt Problems can be implemented, and our Promptly tool currently makes a number of trade-offs: the problem must be solved by a single prompt and dialogue with the model is not allowed, it does not allow students to edit the code that is generated by the LLM, and it evaluates only a single response from the LLM at a time rather than generate and evaluate multiple responses. We believe this provides a suitable experience for introductory level students, but many different variations are possible and should be explored – including letting students engage in dialogue with the LLM and providing the ability to edit the code that is generated. Another particularly interesting variation of Prompt Problems is that instead of representing problems as inputs and outputs, as we have done, students could be presented with a code fragment and tasked with crafting a prompt that generates functionally equivalent code. Such a variation combines aspects of code comprehension with prompt design.

Finally, since prompt creation is a relatively new kind of task, it may be difficult for instructors to have an intuition for how difficult a particular Prompt Problem will be or when to utilize these types of problems. By emphasizing problem solving over syntax, it may make it possible to introduce more complex problems sooner in a course. Future work should explore more rigorously how best to integrate Prompt Problems alongside current teaching practices.

# 7 CONCLUSION

We present a novel pedagogical approach, known as 'Prompt Problems', designed to help students learn how to craft effective prompts for generating code using large language models (LLMs). We report our initial experiences deploying Prompt Problems to students for the first time using a novel tool we have developed, Promptly.

We found that most students were able to solve Prompt Problems in just a few attempts, although some required 20 attempts or more, and that a very wide variety of prompts were constructed. For the most part, students reported very positive experiences solving Prompt Problems, and valued the exposure to new programming constructs and the enhancement of problem-solving skills. However, a small number of students reported some hesitation about automated code generation, and a few even expressed anxiety about the future when seeing how powerful AI code-generating models can be. Future work should investigate different variations of the approach we have described, and explore the right time to introduce students to the concept of prompt-based code generation.

# ACKNOWLEDGMENTS

We are grateful for the grant from the Ulla Tuominen Foundation to Juho Leinonen.

# REFERENCES

[1] Joe Michael Allen, Kelly Downey, Kris Miller, Alex Daniel Edgcomb, and Frank Vahid. 2019. Many Small Programs in CS1: Usage Analysis from Multiple Universities. In *2019 ASEE Annual Conference & Exposition*. ASEE Conferences, Tampa, Florida, 1–13. https://peer.asee.org/33084.
[2] Hannah McLean Babe, Sydney Nguyen, Yangtian Zi, Arjun Guha, Molly Q Feldman, and Carolyn Jane Anderson. 2023. StudentEval: A Benchmark of Student-Written Prompts for Large Language Models of Code. arXiv:2306.04556 [cs.LG]
[3] 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* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[4] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19)*. ACM, NY, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[5] Bruno Pereira Cipriano and Pedro Alves. 2023. GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, NY, USA, 61–67. https://doi.org/10.1145/3587102.3588814

[6] Paul Denny, Brett A. Becker, Juho Leinonen, and James Prather. 2023. Chat Overflow: Artificially Intelligent Models for Computing Education - RenAIssance or ApocAIypse?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, NY, USA, 3–4. https://doi.org/10.1145/3587102.3588773

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

[8] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: Supporting Student-Driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, NY, USA, 471–476. https://doi.org/10.1145/1953163.1953299

[9] 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:2306.02608 [cs.CY]

[10] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Research on Parsons Problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE'20)*. Association for Computing Machinery, NY, USA, 195–202. https://doi.org/10.1145/3373165.3373187

[11] Barbara J. Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S. Miller, Briana B. Morrison, Janice L. Pearce, and Susan H. Rodger. 2022. Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. Association for Computing Machinery, NY, USA, 191–234. https://doi.org/10.1145/3571785.3574127

[12] Tony Haoran Feng, Paul Denny, Burkhard C. Wünsche, Andrew Luxton-Reilly, and Steffan Hooper. 2024. More Than Meets the AI: Evaluating the Performance of GPT-4 on Computer Graphics Assessment Questions. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3636243.3636263

[13] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. ACM, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863

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

[15] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Proceedings of the 26th Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '24)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3636243.3636252

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

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

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

[19] 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* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, NY, USA, 563–569. https://doi.org/10.1145/3545945.3569770

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

[21] Stephen MacNeil, Joanne Kim, Juho Leinonen, Paul Denny, Seth Bernstein, Brett A. Becker, Michel Wermelinger, Arto Hellas, Andrew Tran, Sami Sarsa, James Prather, and Viraj Kumar. 2023. The Implications of Large Language Models for CS Teachers and Students. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, NY, USA, 1255. https://doi.org/10.1145/3545947.3573358

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

[23] Kamil Malinka, Martin Peresíni, Anton Firc, Ondrej Hujnák, and Filip Janus. 2023. On the Educational Impact of ChatGPT: Is Artificial Intelligence Ready to Obtain a University Degree?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, NY, USA, 47–53. https://doi.org/10.1145/3587102.3588827

[24] Leo Porter and Daniel Zingaro. 2023. *Learn AI-Assisted Python Programming: With Github Copilot and ChatGPT.* Manning, Shelter Island, NY.

[25] James Prather, Paul Denny, Juho Leinonen, Brett Becker, et al. 2023. The Robots are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education* (Turku, Finland) *(ITiCSE-WGR '23)*. ACM, NY, USA.

[26] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That It Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (Nov 2023), 31 pages. https://doi.org/10.1145/3617367

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

[28] Advait Sarkar. 2023. Will Code Remain a Relevant User Interface for End-User Programming with Generative AI Models?. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Cascais, Portugal) *(Onward! 2023)*. Association for Computing Machinery, New York, NY, USA, 153–167. https://doi.org/10.1145/3622758.3622882

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

[30] Leonard Tang, Elizabeth Ke, Nikhil Singh, Bo Feng, Derek Austin, Nakul Verma, and Iddo Drori. 2022. Solving Probability And Statistics Problems By Probabilistic Program Synthesis At Human Level And Predicting Solvability. In *Artificial Intelligence in Education: 23rd International Conference, AIED 2022, Durham, UK, July 27–31, 2022, Proceedings, Part II* (Durham, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 612–615. https://doi.org/10.1007/978-3-031-11647-6_127

[31] Matti Tedre and Henriikka Vartiainen. 2023. K-12 Computing Education for the AI Era: From Data Literacy to Data Agency. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, NY, USA, 1–2. https://doi.org/10.1145/3587102.3593796