

# Automating Personalized Parsons Problems with Customized Contexts and Concepts

Andre del Carpio Gutierrez

University of Auckland  
Auckland, New Zealand  
adel868@aucklanduni.ac.nz

Paul Denny

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

Andrew Luxton-Reilly

University of Auckland  
Auckland, New Zealand  
andrew@cs.auckland.ac.nz

## ABSTRACT

Parsons problems provide useful scaffolding for introductory programming students learning to write code. However, generating large numbers of high-quality Parsons problems that appeal to the diverse range of interests in a typical introductory course is a significant challenge for educators. Large language models (LLMs) may offer a solution, by allowing students to produce on-demand Parsons problems for topics covering the breadth of the introductory programming curriculum, and targeting thematic contexts that align with their personal interests. In this paper, we introduce PuzzleMakerPy, an educational tool that uses an LLM to generate unlimited contextualized drag-and-drop programming exercises in the form of Parsons Problems, which introductory programmers can use as a supplemental learning resource. We evaluated PuzzleMakerPy by deploying it in a large introductory programming course, and found that the ability to personalize the contextual framing used in problem descriptions was highly engaging for students, and being able to customize the programming topics was reported as being useful for their learning.

## CCS CONCEPTS

- Social and professional topics → Computing education.

## KEYWORDS

CS1, Large language models, Parsons problems, CS education tools, Personalized learning

### ACM Reference Format:

Andre del Carpio Gutierrez, Paul Denny, and Andrew Luxton-Reilly. 2024. Automating Personalized Parsons Problems with Customized Contexts and Concepts. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3649217.3653568>

## 1 INTRODUCTION

The primary goal for students in introductory programming courses has traditionally been to learn how to write code. Beginners often find it challenging and time-consuming to write code from scratch [1, 31], and thus effective activities to scaffold code writing are valuable. Parsons problems have become one of the most widely studied and popular scaffolding tasks in introductory courses [10].

*ITiCSE 2024, July 8–10, 2024, Milan, Italy*

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*, <https://doi.org/10.1145/3649217.3653568>.

They are programming puzzles where students rearrange incorrectly ordered code fragments into a correct working program [28]. Although Parsons problems have several benefits for students [10], it is time-consuming for educators to create large numbers of these educational resources [27]. In addition, we believe that the learning experience is likely to be improved when students have the autonomy to personalize their learning resources by selecting problem contexts that align with their interests [23]. Large language models (LLMs) may help with automating the creation of these exercises, as they have been shown capable of producing thematic exercises (i.e., generating a context for a given problem) and their associated code solutions [32].

In this research, we describe PuzzleMakerPy, an educational tool that uses an LLM (OpenAI's GPT-3.5) to automatically create an unlimited number of personalized Parsons problems in which the context and programming concepts can be tailored to individual students. We deployed PuzzleMakerPy in a large introductory programming course, and evaluated its use guided by the following research questions:

- **RQ1:** *How do introductory programming students interact with the PuzzleMakerPy educational tool?*
- **RQ2:** *What attitudes emerge from their engagement with the tool?*

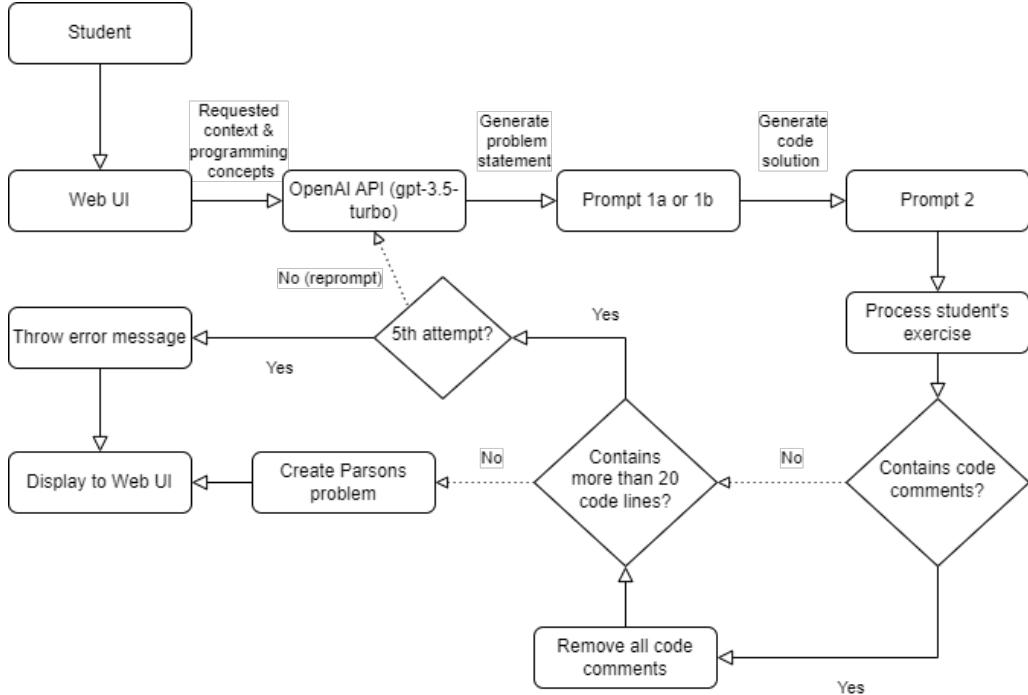
We found that PuzzleMakerPy was well-received by students, who reported the ability to personalize Parsons problems via contexts and programming concepts enjoyable and useful for their learning. Furthermore, we found that students used a wide range of the provided contexts available to them in the tool, and would often prescribe their own contexts. We also found that students used all provided programming concepts but never entered their own, indicating that a reasonably sized list that covers typical introductory programming curricula may suffice for their learning.

## 2 RELATED WORK

Extensive prior work exists on both the use and integration of LLMs in programming education and on Parsons problems.

### 2.1 Large Language Models in Programming Education

The integration of large language models (LLMs) into programming education has recently gained significant attention [7, 30]. A growing body of literature now documents the capabilities of LLMs for solving a range of tasks relevant in computing courses, including generating solutions to programming problems [5, 14, 17, 33], test suite creation [3], detecting bugs [26], and explaining both code [22, 32] and error messages [24, 29]. LLM-powered tools are



**Figure 1: A diagram illustrating the architecture and information flow of PuzzleMakerPy.**

also entering computing classrooms as digital teaching assistants [25], programming support tools [21], and tools to support the development of skills in prompt creation and engineering [6, 8]. Researchers have also explored the application of LLMs to generate learning resources for students. Sarsa et al. explored the use of a pioneering LLM, Codex, for generating novel and sensible programming exercises that could be customised according to specified programming concepts and contextual themes [32]. Although much of the content generated using their few-shot approach was accurate, they suggested there was a need for oversight from instructors to ensure content quality. Similar work on resource generation for programming courses has focused on multiple-choice questions [9] and worked examples [20].

## 2.2 Parsons Problems

Numerous variations of Parsons problems have been explored, including puzzles that are adaptive [11, 18], Parsons problems with or without distractors [16], single line puzzles related to regular expressions or SQL (micro) [30], optional code lines [13], and faded Parsons problems (with incomplete code fragments that need to be filled) [15, 34]. Parsons problems are known to provide many learning benefits for students such as improved learning engagement [28], the ability to learn from immediate feedback [10, 12], the potential to reduce cognitive load [35], and to result in similar learning gains when compared to writing code from scratch [13, 28, 35].

In the current paper, we explore the automatic generation of Parsons problems. Most similar to this work is the research of Hou et al., who present the CodeTailor tool, a system that uses LLMs to create personalized Parsons puzzles [19]. In a study with 18

novice programmers, they found that most students preferred to use CodeTailor over receiving AI-generated solutions to support their learning. We previously evaluated the quality of LLM-generated programming exercises spanning a broad range of contexts and programming concepts [4]. The generated exercises included high-quality problem statements and code solutions, providing the necessary components to create Parsons problems for introductory programmers.

## 3 TOOL DESIGN AND IMPLEMENTATION

The architecture (see Figure 1), design choices, technologies, and prompts used for PuzzleMakerPy are detailed in this section.

When students use PuzzleMakerPy, they are asked to personalize their problem by selecting a context and a set of programming concepts (see Figure 2). After generating the problem, students are provided with a problem description (see Figure 3) and can use a typical drag-and-drop interface to develop their solutions (see Figure 4). Feedback on the submitted code solutions is also provided, as shown in Figure 5.

### 3.1 Filtering Automated Content

Figure 1 provides an overview of the architecture of PuzzleMakerPy, which is designed to reliably create high-quality Parsons problems. The first design decision was limiting the number of code fragments in generated solutions through our zero-shot prompts (see Figure 6) and by reprompting only if solutions contained more than 20 lines of code. Reprompting for an exercise that fails to meet the conditions is performed five times consecutively before the student is asked to retry. This limit ensures that students are not stuck

**Step 1: Select a Context**

Cats

Or type your own

**Step 2: Select Programming Concepts (maximum of 3)**

- Variables
- Strings
- Arithmetic operators
- Selection statements
- Loops
- Lists
- Dictionaries
- File handling & I/O

Or type your own (must be comma separated ",")

**Figure 2:** An image illustrating how a student would interact with PuzzleMakerPy when selecting their context and programming concepts to create a programming exercise.

**Step 4: Read the problem description**

Hide Problem Description

**Problem:**

Write a program to calculate the total number of days a cat has lived based on its age in years and months. Consider that each month has 30 days. The program should display the result in the format "The cat has lived for X days."

Programming Concepts: strings, arithmetic operators, selection statements.

**Figure 3:** An image illustrating PuzzleMakerPy's interface for generated problem statements.

**Step 5: Drag-and-drop your solution**

Drag from here

```
total_days = (age_years * 365) + (age_months * 30)
print("The cat has lived for", total_days, "days.")
```

Construct your solution here

```
age_months = int(input("Enter the cat's age in months: "))
age_years = int(input("Enter the cat's age in years: "))
```

**Step 6: Verify your solution**

**Figure 4:** An image illustrating PuzzleMakerPy's drag-and-drop interface for code solutions.

waiting, and reduces API costs if the LLM repeatedly fails to create the exercise. Another design choice included removing all code comments from exercises, as Parsons problems blocks typically do not contain comments. Finally, to reduce the likelihood of producing undesirable responses, our zero-shot prompts excluded various programming concepts in generated solutions, as these did not

The highlighted fragment 4 belongs to a wrong block (i.e. indentation). Construct your solution here

```
age_years = int(input("Enter the cat's age in years: "))

age_months = int(input("Enter the cat's age in months: "))

total_days = (age_years * 365) + (age_months * 30)

print("The cat has lived for", total_days, "days.")
```

**Figure 5:** An image illustrating how PuzzleMakerPy provides feedback for a solution containing incorrect indentation.

**Prompt 1a (problem description when a student requests for a context):**

Generate a problem in Python that only includes a problem statement with a context of '[context requested by a student goes here]', and the programming concepts should be '[programming concepts requested by a student goes here]'. The solution associated with the problem should not exceed ten lines of code and should be limited to only a single function at most. The problem cannot use while(true), or any break statements or exceptions (try, catch blocks). Do not include information on the constraints I have defined in the problem statement. Only provide the problem statement as the output.

**Prompt 1b (problem description when a student requests for no context):**

Generate a problem in Python that only includes a problem statement with the following programming concepts: '[programming concepts requested by a student goes here]'. Do not apply a context to the exercise. The solution associated with the problem should not exceed ten lines of code and should be limited to only a single function at most. The problem cannot use while(true), or any break statements or exceptions (try, catch blocks). Do not include information on the constraints I have defined in the problem statement. Only provide the problem statement as the output.

**Prompt 2 (code solution):**

[problem description outputted from either prompt 1a or 1b]

Only provide code solutions based on the problem description above. Do not explain the solution or add any extra detail to the output; only provide the code solution.

**Figure 6:** All three zero-shot prompts used for generating complete programming exercises within PuzzleMakerPy. Prompt 1a or Prompt 1b is used to generate the problem description, while Prompt 2 is solely used to generate code solutions.

align with the learning objectives specific to the course that the tool was deployed in, and it may have confused students. We also included prompts to ensure that problem statements did not contain extra unnecessary text.

### 3.2 Technologies

PuzzleMakerPy was developed using Flask (a micro web framework written in Python) and standard web technologies such as HTML, CSS, JavaScript, and jQuery. An external library that contributed to the creation of this educational tool is the open source *js-parsons* library<sup>1</sup>.

The LLM used for prompting all programming exercises in PuzzleMakerPy was *gpt-3.5-turbo*. Multiple zero-shot prompts were employed for exercise generation. The first prompt focused on generating the problem description based on the context or thematic keyword provided by a student, denoted as either Prompt 1a or Prompt 1b. Prompt 1a handled requests for exercises that required a context, while Prompt 1b was used for exercises without a context. The second prompt (Prompt 2) handled the generation of code solutions, and this prompt was supplemented with the problem description obtained from either Prompt 1a or Prompt 1b. To observe the three zero-shot prompts for generating programming exercises requested by students, refer to Figure 6.

### 3.3 Contexts and Programming Concepts

Students were allowed to select contexts and programming concepts for their generated programs (see Figure 2). To cover a wide range of student interests, 20 contexts (see Table 1) were integrated into PuzzleMakerPy through a drop-down menu. In addition, students could choose to include either no context, type a specific context, or (by selecting a ‘Surprise me’ option) use a random context selected from the topics of the Dewey Decimal Classification System (approximately 1000 topics). Introducing this level of flexibility is likely to align with students’ interests, allowing them to discover enjoyable and engaging programming exercises.

Similarly, for programming concepts, students were given the choice to select a maximum of 3 concepts for generated solutions using checkboxes, as shown in Figure 2. We believe that being able to combine multiple concepts for generated solutions would lead to more useful learning resources for students. For the programming concepts used (see Table 1), students could also type their own concepts to be included before an exercise is created. However, unlike requesting exercises without context, students were not allowed to create exercises without programming concepts.

**Table 1: List of contexts and programming concepts used in PuzzleMakerPy**

Contexts: 20
Amusement Park; Animals; Aquarium; Basketball; Cooking; Film; Fishing; Gardening; Mental Health; Modern Gaming; Music; Olympics; Pets; Relationships; Restaurant; Rugby; Social Media; Sports; Streaming Services; Virtual Reality
Programming Concepts: 8
Arithmetic operators; Dictionaries; File handling & I/O; Lists; Loops; Selection statements (if/else, etc.); Strings; Variables

## 4 EVALUATION

PuzzleMakerPy was deployed approximately halfway through an introductory programming course (CS1) with 423 students. Using

<sup>1</sup><https://github.com/js-parsons/js-parsons>

the tool was optional for students, but they were encouraged to use it to revise their code-writing skills before a mid-semester test. Students in the course had not previously been taught using Parsons problems.

As use of the tool was anonymous, each problem request was logged independently, and so we can only report the number of attempts rather than the number of students using the tool. We logged elements of student behavior and asked them to complete an optional questionnaire.

### 4.1 Student Requested Keywords

Students had various options for selecting contexts and programming concepts before generating exercises, either from predefined lists (see Table 1) or by typing their own. We collected logs of use from PuzzleMakerPy and report the top 20 student-requested contexts in Table 2. Additionally, the frequency of student-requested programming concepts is shown in Table 3.

**Table 2: Top 20 student-requested contexts in generated questions (out of 236).** ‘None’ refers to no-context, ‘Surprise Me’ is a random context extracted from the Dewey Decimal System, and ‘Custom’ is a context typed by a student.

Context Requested	Number of questions
Animals	33
Music	22
Custom	20
Basketball	17
Cats	17
Amusement Park	14
Mental Health	11
Modern Gaming	9
Sports	9
Film	8
Fishing	8
None	8
Relationships	8
Cooking	5
Olympics	5
Pets	5
Surprise Me	5
Gardening	4
Restaurant	4
Social Media	3

**Table 3: The frequency of student-requested programming concepts in generated questions (out of 236)**

Programming Concept Requested	Number of questions
Loops	110
Variables	106
Strings	78
Lists	77
Dictionaries	73
Arithmetic Operators	32
File Handling & I/O	30
Selection Statements	17

For requested contexts, ‘animals’ were the most popular (33 questions), and relatively few questions were generated with ‘none’ (8

questions), and even fewer using the ‘surprise me’ (5 questions) feature. Many questions were also generated using ‘custom’ contexts (20 questions) which were entered by students, indicating there is value in including this option despite the extensive list of contexts provided.

For programming concepts requested by students, ‘loops’ were the most prevalent (110 questions), with ‘variables’ a close second (106 questions). Variables may appear high on the list as they are commonly used in programs and are a generally well-understood concept. Students used all of the programming concepts offered, although the distribution was not even, but surprisingly no student entered their own programming concepts. This suggests that using a small list focused on the key concepts relevant to the course is likely sufficient.

In summary, student behavior shows a wide range of contexts being used, with few opting not to use contextualization. Similarly, students used all of the predefined programming concepts when generating questions, although no questions were generated with student-entered topics.

## 4.2 Student satisfaction

After completing exercises using the tool, students were invited to respond to a short questionnaire. Four Likert-scale questions asked students about their preferences, followed by four open-ended questions that asked students to reflect on their experiences. The questions are shown in Tables 4 and 5.

**Table 4: Likert scale survey questions and results ( $n = 18$ )**

**Abbreviations:** SD = Strongly Disagree, D = Disagree, N = Neutral, A = Agree, SA = Strongly Agree

Question	SD	D	N	A	SA
Q1. Customizing the context of a programming question was interesting/enjoyable.	0%	0%	0%	50%	50%
Q2. Customizing programming concepts is valuable for improving my own programming skills.	0%	0%	11%	44%	44%
Q3. Solving customized/personalized drag-and-drop programming questions was useful for my learning.	0%	6%	11%	39%	44%
Q4. Generating unlimited personalized drag-and-drop programming questions was useful for my learning.	0%	0%	22%	28%	50%

## 4.3 Likert Scale Survey Responses

A total of 18 students engaged with the Likert scale survey questions and their responses are summarized in Table 4.

Students generally agreed that the ability to select their own context (Q1) was enjoyable. This supports the idea that personalizing Parsons problems was enjoyable, likely enhancing the learning experience of introductory programmers. Including various contexts contributes to positive feedback, suggesting that limiting contexts in a similar tool may not yield the same level of positive response, as it may fail to align well with the diverse range of student interests.

**Table 5: Open-ended survey questions**

n	Question	Optional?
18	Q1. Why do you (or do not) find the customization of context in programming questions interesting/enjoyable? You can potentially describe any instances you have encountered with the tool to help support your answer.	No
18	Q2. Describe the usefulness of being able to select programming concepts in this tool. You can potentially describe any instances you have encountered with the tool to help support your answer.	No
7	Q3. Any overall thoughts or comments about the customization/personalization of programming questions can be included here.	Yes
62	Q4. Do you have any other feedback you would like to provide about PuzzleMaker?	Yes

Additionally, they responded positively regarding the value of selecting programming concepts (Q2) for improving their programming skills. Moreover, students were very positive about the usefulness of solving personalized Parsons problems for their learning. Furthermore, they were very positive about generating an unlimited number of personalized Parsons problems being useful for their learning.

The tool allows students to combine contexts and programming concepts in exercises, likely maintaining student enjoyment and exposing them to relevant learning materials. In response to our second research question (RQ2), the findings suggest that students generally responded positively to the tool, implying that they find contextualized programming exercises in the form of drag-and-drop (Parsons problems) interesting and useful for their learning.

## 4.4 Open-ended Survey Responses

The data from four open-ended questions (see Table 5) were explored using thematic analysis. Relevant responses were extracted as codes and grouped into themes by the first author [2].

**4.4.1 Customisation of Contexts.** The question relevant to the inclusion of contexts (Q1) had four themes found within student responses (enjoyment, self-expression, value, and counterproductivity).

The first theme, **enjoyability**, captures how context makes solving small programming exercises more pleasant than the typical experience of working with programming exercises devoid of context. Responses that illustrate this theme were “*It makes the learning process more enjoyable and interesting if the context in which I am trying to solve the problem is something that interests me instead of some random generic problem.*” and “*It’s easier to work with something you’re interested in*”.

The second theme acknowledged that generating questions via the tool enabled **self-expression**. This theme describes students being given the creative freedom and control to create programming questions that align with their interests, with responses such as “*I can come up with programming questions involving topics that I like and stuff I have as a hobby*”, “*Because we can choose what our code is about*.”, and “*because it gives us a sense of ‘oh we created this problem’*”.

The third theme indicated that students believed that context was **valuable** for learning. Responses associated with this theme were “*It can be helpful to explain the context to understand the problem more and find what would be the best structure for the code to resolve the problem*” and “*Was helpful to relate puzzles to different concepts*”.

The final theme was more negative and encompassed the view that contexts could be **counterproductive**. Specifically, contexts may introduce added difficulty into the interpretability of problem statements, as witnessed by the comment “*unnecessary context might be distracting from the question at hand*”.

**4.4.2 Customisation of Programming Concepts.** In the open-ended responses regarding the value of programming concepts (Q2), three themes emerged from student feedback. The first theme focuses on students’ **learning opportunity**, for repeated practice of programming concepts. Practice questions may serve as a way to enhance understanding or serve as a general refresher for students, as indicated by the comments “*It is useful because it enables targeted revision*”, “*It lets us practice concepts that we may be struggling with, or also just help solidify anything that we have just learned.*”, and “*it helps as I can focus on the areas that I feel less confident in and work through them at my own pace*”.

The second theme was **exposure**. This theme highlights an opportunity for introductory programmers to encounter new concepts or ideas for solving problems they may not have seen or thought about before. Comments aligned with this theme included “*it is nice to be able to see how a question should be answered if you are struggling with how to code something. examples are incredible at helping you see what you should do and where you can implement the same things in your code.*”, and “*I can experience more problems and further my programming skills and resolve problems more efficiently, if i ever encounter the same problem to recall what I should be doing*”.

The final theme for this question was **limited**, describing some of the tool’s inflexibility with applying code solutions. A common observation is that problems often have multiple solutions, as indicated by the comment “*I didn’t like their solution - I wanted to have the ability to provide my own solution.*”. Additionally, the tool integrated programming concepts into code solutions but failed to do something unique with them, as suggested in the following comment “*I think this tool would be useful if I was struggling to understand a specific concept in programming and this tool would help in explaining the concept in an interesting way*”.

**4.4.3 Personalisation of Parsons Problems.** There were relatively few responses to Q3, resulting in only two themes. The first theme was **useful**, indicating how students saw personalized questions as a practical additional learning resource reinforcing their understanding of programming knowledge. Comments supporting this theme included “*I thought it was great overall, a really good tool very helpful for learning and quite fun. Nice to use the drag and drop for understanding the structure of your code.*”, and “*I would use that to better understand how I should structure my code and what I should write in my code to get the output that I want.*”. In contrast to the **useful** theme, the final theme was **undemanding**, describing the questions as lacking variety and not reflecting the difficulty of real-world programming problems. Responses reflecting this idea included “*The questions seemed to be quite easy, maybe try make them harder.*”, and “*Will it be possible to incorporate challenging*

*questions that can be encountered in daily programming? Or show different ways to solve the problem.*”. One student also remarked “*Needs more complex questions exam style*”, highlighting a mismatch between the Parsons problems format and the typical code writing format used in exams.

**4.4.4 Other Tool Remarks.** Three themes were identified from student responses to the final optional open-ended question (Q4). The first theme was **tool issues**, which included the tool’s appearance, functionality, and the limitations of basic Parsons problems compared to writing code from scratch. Comments reflecting this theme include “*Upgrade the UI, maybe adding problems in other programming languages if possible*”, “*I wish the site would behave more like coderunner where it can check the code that I wrote instead of the drag-and-drop format its in right now*”, “*I think rather than drag and drop, it should make the user to actually write the code - similar to lab exercises.*”, and “*I also feel like it’s less useful as revision than trying to write the code yourself*”.

The second theme was **learning opportunity** and included comments such as “*Personally I think this is a useful tool as some people like myself can’t really just make up our own puzzles to practice on.*”, “*It’s helpful for visualizing and practicing code structure.*”, and “*I found it somewhat helpful, it is a nice alternative to study*”.

The last theme was students finding **misleading** code solutions that do not match what they were explicitly taught in their introductory programming courses. Comments illustrating this theme were “*Mine didn’t seem to have the code written as best practice*”, and “*Some code I had not used before so it was bit confusing*”.

## 5 CONCLUSION

We developed an educational tool, PuzzleMakerPy, that leverages a large language model for automating the generation of Parsons problems with customizable programming concepts and thematic contexts. We deployed PuzzleMakerPy in a large introductory programming course, finding that students viewed the ability to customize Parsons problems with contexts as enjoyable, and reported the ability to select programming concepts as useful for their learning. We found that students were enthusiastic about being able to create programming questions that aligned with their interests. They engaged with a wide variety of available contexts, some even typing their own, and few requested Parsons problems without context. Students also made use of the programming concepts provided to them in a small predefined list but did not choose to type their own. Overall, our findings highlight the strengths and benefits of PuzzleMakerPy, particularly how the contextualization of Parsons Problems can be enjoyable for introductory programmers while maintaining learning benefits by allowing customization of programming topics. Future work should continue to explore the potential for automatically generating personalized learning resources for students, especially by leveraging large language models.

## 6 ACKNOWLEDGEMENTS

This study was conducted with the ethics approval of *University of Auckland Human Participants Ethics Committee (UAHPEC)*, with reference *UAHPEC25279*.

## REFERENCES

- [1] Klara Benda, Amy Bruckman, and Mark Guzdial. 2012. When Life and Learning Do Not Fit: Challenges of Workload and Communication in Introductory Computer Science Online. *ACM Transactions on Computing Education* 12, 4 (Nov. 2012), 15:1–15:38. <https://doi.org/10.1145/2382564.2382567>
- [2] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- [3] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhi Chen. 2022. CodeT: Code Generation with Generated Tests. <https://doi.org/10.48550/arXiv.2207.10397>
- [4] Andre Del Carpio Gutierrez, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating Automatically Generated Contextualised Programming Exercises. In *Proc of the 55th ACM Tech. Sym. on Comp. Sci. Education V. 1 (SIGCSE 2024)*. ACM, New York, NY, USA, 289–295. <https://doi.org/10.1145/3626252.3630863>
- [5] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proc of the 54th ACM Tech. Sym. on Comp. Sci. Education V. 1 (SIGCSE 2023)*. ACM, New York, NY, USA, 1136–1142. <https://doi.org/10.1145/3545945.3569823>
- [6] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Theziry 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)*. ACM, New York, NY, USA, 296–302. <https://doi.org/10.1145/3626252.3630999>
- [7] 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. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (Jan 2024), 56–67. <https://doi.org/10.1145/3624720>
- [8] Paul Denny, David H. Smith, Max Fowler, James Prather, Brett A. Becker, and Juho Leinonen. 2024. Explaining Code with a Purpose: An Integrated Approach for Developing Code Comprehension and Prompting Skills. <https://doi.org/10.48550/arXiv.2403.06050>
- [9] Jacob Dougherty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, Arav Agarwal, Christopher Bogart, Eric Keylor, Can Kultur, Jaromir Savelka, and Majd Sakr. 2024. A Comparative Study of AI-Generated (GPT-4) and Human-crafted MCQs in Programming Education. In *Proc of the 26th Australasian Comp. Ed. Conf. (ACE '24)*. ACM, New York, USA, 114–123. <https://doi.org/10.1145/3636243.3636256>
- [10] 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*. ACM, Dublin Ireland, 191–234. <https://doi.org/10.1145/3571785.3574127>
- [11] Barbara J. Ericson, James D. Foley, and Jochen Rick. 2018. Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18)*. ACM, New York, NY, USA, 60–68. <https://doi.org/10.1145/3230977.3231000>
- [12] Barbara J. Ericson, Mark J. Guzdial, and Briana B. Morrison. 2015. Analysis of Interactive Features Designed to Enhance Learning in an Ebook. In *Proceedings of the eleventh annual International Conf. on Int. Computing Education Research*. ACM, Omaha Nebraska USA, 169–178. <https://doi.org/10.1145/2787622.2787731>
- [13] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. 2017. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. Koli Finland, 20–29. <https://doi.org/10.1145/3141880.3141895>
- [14] 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 *Proc of the 24th Australasian Comp Ed Conference (ACE '22)*. ACM, NY, USA, 10–19. <https://doi.org/10.1145/3511861.3511863>
- [15] Flynn Fromont, Hiruna Jayamanne, and Paul Denny. 2023. Exploring the Difficulty of Faded Parsons Problems for Programming Education. In *Proc of the 25th Australasian Computing Education Conference (ACE '23)*. ACM, NY, USA, 113–122. <https://doi.org/10.1145/3576123.3576136>
- [16] Kyle James Harms, Jason Chen, and Caitlin L. Kelleher. 2016. Distractors in Parsons Problems Decrease Learning Efficiency for Young Novice Programmers. In *Proc of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, NY, USA, 241–250. <https://doi.org/10.1145/2960310.2960314>
- [17] Irene Hou, Owen Man, Sophia Mettille, Sebastian Gutierrez, Kenneth Angelikas, and Stephen MacNeil. 2024. More Robots are Coming: Large Multimodal Models (ChatGPT) can Solve Visually Diverse Images of Parsons Problems. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. ACM, New York, NY, USA, 29–38. <https://doi.org/10.1145/3636243.3636247>
- [18] Xinying Hou, Barbara Jane Ericson, and Xu Wang. 2022. Using Adaptive Parsons Problems to Scaffold Write-Code Problems. In *Proceedings of the 2022 ACM Conference on International Computing Education Research V. 1 (ICER '22)*. ACM, Lugano and Virtual Event Switzerland, 15–26. <https://doi.org/10.1145/3501385.3543977>
- [19] Xinying Hou, Zihan Wu, Xu Wang, and Barbara J. Ericson. 2024. CodeTailor: Personalized Parsons Puzzles are Preferred Over AI-Generated Solutions to Support Learning. <https://doi.org/10.48550/arXiv.2401.12125>
- [20] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *Proc of the 26th Australasian Comp Ed Conference (ACE '24)*. ACM, NY, USA, 77–86. <https://doi.org/10.1145/3636243.3636252>
- [21] Majeed Kazemitaabari, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weinrop, 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 (CHI '23)*. ACM, New York, NY, USA, Article 455, 23 pages. <https://doi.org/10.1145/3544548.3580919>
- [22] 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 (ITiCSE 2023)*. ACM, New York, NY, USA, 124–130. <https://doi.org/10.1145/3587102.3588785>
- [23] Juho Leinonen, Paul Denny, and Jacqueline Whalley. 2021. Exploring the Effects of Contextualized Problem Descriptions on Problem Solving. In *Proc of the 23rd Australasian Computing Education Conference (ACE '21)*. ACM, New York, NY, USA, 30–39. <https://doi.org/10.1145/3441636.3442302>
- [24] 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*. ACM, Toronto ON Canada, 563–569. <https://doi.org/10.1145/3545945.3569770>
- [25] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2024. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *Proc of the 23rd Koli Calling Int. Conf. on Computing Education Research (Koli Calling '23)*. ACM, NY, USA, Article 8, 11 pages. <https://doi.org/10.1145/3631802.3631830>
- [26] Stephen Macneil, Paul Denny, Andrew Tran, Juho Leinonen, Seth Bernstein, Arto Hellas, Sami Sarsa, and Joanne Kim. 2024. Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. ACM, New York, NY, USA, 11–18. <https://doi.org/10.1145/3636243.3636245>
- [27] Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. 2023. What Is Your Biggest Pain Point? An Investigation of CS Instructor Obstacles, Workarounds, and Desires. In *Proc of the 54th ACM Tech. Sym. on Comp. Sci. Education V. 1 (SIGCSE 2023)*. ACM, NY, USA, 291–297. <https://doi.org/10.1145/3545945.3569816>
- [28] Dale Parsons and Patricia Haden. 2006. Person's programming puzzles: a fun and effective learning tool for first programming courses. In *Proc 8th Australasian Conf. on Comp. Ed. - Vol. 52 (ACE '06)*. Australian Comp. Soc., AUS, 157–163.
- [29] Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. *arXiv preprint arXiv:2302.04662* (2023). <https://arxiv.org/abs/2302.04662>
- [30] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proc of the 2023 Working Group Reports on Innovation and Technology in CS Education (ITiCSE-WGR '23)*. ACM, New York, NY, USA, 108–159. <https://doi.org/10.1145/3623762.3633499>
- [31] Yizhou Qian and James Lehman. 2018. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education* 18, 1 (March 2018), 1–24. <https://doi.org/10.1145/3077618>
- [32] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proc of the 2022 ACM Conf. on Int. Comp. Ed. Research - Volume 1 (ICER '22)*, Vol. 1. ACM, NY, USA, 27–43. <https://doi.org/10.1145/3501385.3543957>
- [33] Jaromir Savelka, Arav Agarwal, Christopher Bogart, Yifan Song, and Majd Sakr. 2023. 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 (ITiCSE 2023)*. ACM, New York, NY, USA, 117–123. <https://doi.org/10.1145/3587102.3588792>
- [34] Nathaniel Weinman, Armando Fox, and Marti A. Hearst. 2021. Improving Instruction of Programming Patterns with Faded Parsons Problems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–4. <https://doi.org/10.1145/3411764.3445228>
- [35] Rui Zhi, Min Chi, Tiffany Barnes, and Thomas W. Price. 2019. Evaluating the Effectiveness of Parsons Problems for Block-based Programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. ACM, Toronto ON Canada, 51–59. <https://doi.org/10.1145/3291279.3339419>