CAN WE TRUST AI-GENERATED EDUCATIONAL CONTENT? COMPARATIVE ANALYSIS OF HUMAN AND AI-GENERATED LEARNING RESOURCES

Paul Denny
School of Computer Science
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Hassan Khosravi
Institute for Teaching and Learning Innovation
The University of Queensland
Brisbane, Queensland, Australia
h.khosravi@uq.edu.au

Arto Hellas
Department of Computer Science
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

Juho Leinonen
School of Computer Science
The University of Auckland
Auckland, New Zealand
juho.leinonen@auckland.ac.nz

Sami Sarsa
Department of Computer Science
Aalto University
Espoo, Finland
sami.sarsa@aalto.fi

July 4, 2023

ABSTRACT

As an increasing number of students move to online learning platforms that deliver personalized learning experiences, there is a great need for the production of high-quality educational content. Large language models (LLMs) appear to offer a promising solution to the rapid creation of learning materials at scale, reducing the burden on instructors. In this study, we investigated the potential for LLMs to produce learning resources in an introductory programming context, by comparing the quality of the resources generated by an LLM with those created by students as part of a learnersourcing activity. Using a blind evaluation, students rated the correctness and helpfulness of resources generated by AI and their peers, after both were initially provided with identical exemplars. Our results show that the quality of AI-generated resources, as perceived by students, is equivalent to the quality of resources generated by their peers. This suggests that AI-generated resources may serve as viable supplementary material in certain contexts. Resources generated by LLMs tend to closely mirror the given exemplars, whereas student-generated resources exhibit greater variety in terms of content length and specific syntax features used. The study highlights the need for further research exploring different types of learning resources and a broader range of subject areas, and understanding the long-term impact of AI-generated resources on learning outcomes.

Keywords Learnersourcing · Large language models · Codex · Generative AI · Student-generated resources

1 Introduction

The cornerstone of successful learning communities is ready access to high quality educational resources that enable learners to follow their own learning paths. When paired with a learning management system that keeps track of

progress and behavior, large pools of learning resources allow personalized learning experiences that are tailored to match the needs of every student [1]. However, creating and maintaining such large pools of learning resources can take significant effort. One possibility to ease the burden of creating learning resource pools is the use of learnersourcing, where learners are engaged in the collaborative creation and curation of their own learning resources [2, 3]. Although learnersourcing provides ample benefits to students including access to varied educational content and fostering a shared sense of ownership of learning, there are several well-documented issues including low quality and inaccurate resources and challenges around incentivization [3].

The recent emergence of generative AI tools powered by large language models (LLMs) has given rise to a new approach for creating learning resources. Novel content can be created with LLMs, such that the learning objectives and themes of the generated resources can be directed with examples and prompts [4]. Despite this great promise, very little research exists that rigorously evaluates the quality of LLM-generated content. In particular, to our knowledge there is no comparative assessment of learning resources that are created by students and by LLMs when both are provided with an identical set of initial examples. In this work, we prepared six example learning resources suitable for students in an introductory programming course. We provided these six examples to students to help them get started with a learnersourcing activity, inviting them to create their own similar resources. We also provided the same six resources to an LLM as few-shot examples, prompting it to generate new ones. We present the results of a blind evaluation of the quality of both the student-generated and AI-generated resources. The evaluation involved students rating all learning resources with respect to correctness and helpfulness. In addition to student perceptions of the learning value of the resources, we are also interested in understanding how the content of the resources differ. The following two research questions guide our work.

- **RQ1:** Given identical priming examples, how do student-generated and AI-generated learning resources differ in terms of overall length and presence of syntax features?
- **RQ2:** How do students rate the correctness and helpfulness of learning resources when they are student-generated compared to when they are AI-generated?

2 Related Work

2.1 AI in Content Generation

Large language models (LLMs) have garnered widespread attention in recent years, due to their capability to process and generate text at a very high level of sophistication. This has resulted in considerable interest in educational settings, where LLMs appear poised to revolutionise teaching and assessment practices. In particular, LLMs can be applied to the problem of generating high-quality educational resources, including practice problems [4], explanations [5], and textual summaries. Wang et al. investigated the potential of LLMs for generating questions that could be used in educational settings, and explored a variety of different prompting strategies [6]. They found under certain prompting conditions, the generated questions were indistinguishable to a subject matter expert when compared with human-generated questions. In similar work, Bulathwela et al. argue that automatic question generation via LLMs will play an important role in scaling online education [7]. They present a tool which applies an additional pre-training step involving scientific text documents to an existing pre-trained model and then fine-tune it for question generation. They illustrate that the additional training steps yield higher quality questions, but also note that lack of human evaluation of the generated questions remains an important gap in educational settings. These studies collectively highlight the promising potential of LLMs in automating and personalizing educational tasks, while also emphasizing the need to ensure the quality and relevance of generated content.

However, despite this potential, concerns about the ethical and practical challenges around LLMs have emerged, including concerns around bias in generated content [8], academic integrity concerns [9], and the potential impact on educational research and practice [10]. Yan et al. conducted a systematic literature review to identify the practical and ethical challenges associated with LLMs in education, providing recommendations for future research focused on developing practical, ethical, and human-centered innovations [11]. Similarly, Chiu et al. performed a systematic review on AI in education, identifying opportunities, challenges, and future research directions across learning, teaching, assessment, and administration domains [12]. One of the key challenges highlighted in this review is the lack of relevant learning resources for personalized and adaptive learning, a challenge which LLMs may be able to help address.

Overall, the literature on LLMs in education presents a complex landscape of both opportunities and challenges [13, 14]. The potential of LLMs for automating and personalizing education through content generation is promising, however it is crucial to address ethical, practical, and academic integrity concerns to ensure responsible and effective integration of these technologies into educational systems.

2.2 Large Language Models in Computing Education

Across all educational disciplines, there has been significant recent interest in large language models and the challenges and opportunities they present to educators and students [14]. Within the field of computing education, where code-generation models face additional constraints on the formation of syntactically and semantically valid outputs, there has also been increasing research interest [15, 16]. Computing education focused studies have, to date, mostly explored the performance of code generation models on solving typical introductory programming problems [17, 18]. For instance, Finnie-Ansley et al. published a pair of articles that evaluated the performance of the Codex model on a private database of CS1 and CS2 programming problems from high-stakes end-of-course exams [19, 20]. In both cases, Codex generated solutions that achieved scores that easily surpassed most students in the course. A similar study using a public database of programming problems found that Codex was able to produce correct solutions on its first try approximately 50% of the time, but that with minor modifications to the input prompt this success rate increased to 80% [21]. In programming-related disciplines such as bioinformatics, where scientists are often not expert programmers, Piccolo et al. showed that ChatGPT could solve three-quarters of basic to moderate level programming tasks on its first attempt, rising to 97% if allowed up to 7 attempts [22].

Although receiving less attention to date, there has been increasing interest in exploring the capability of AI-based code generation models for producing learning resources. One of the first examples of this work was a study by Sarsa et al. on using Codex for generating programming exercises and code explanations [4]. They found the generated code explanations to be both thorough and fairly accurate, with 90% of the explanations sufficiently explaining all parts of the code, and observed 70% of the individual line by line explanations to be correct. However, in this prior work, the evaluation was only performed by experts and the resources were never shown or evaluated by students. MacNeil et al. also utilized Codex to generate explanations for short code segments, however these were presented to students by placing the explanations next to the corresponding code within in an online course textbook [23]. Their evaluation was limited to around 50 participants, however the students who did access the code explanations found them useful when they chose to engage with them. Overall, the authors found that the students' engagement was lower than expected and students did not participate in the creation of either the code examples or the accompanying explanations.

2.3 Learnersourcing

Learnersourcing is a form of crowdsourcing, where the participants in the crowd are learners [2]. Typically, learnersourcing is actualized by having students on a course do some task as part of their course work. One common learnersourcing use case is having students create exercises of their own, often in addition to completing traditional exercises [24]. This process usually involves students also reviewing the content produced by their peers. The increasing attention towards learnersourcing has led to new research and methodologies aimed at shaping the development of learnersourcing tools and guiding related research activities. In their study, Khosravi et al. [25] reflect upon their experience in constructing an adaptive learnersourcing platform, sharing a series of data-driven insights for those in development and research roles. Subsequenty, Singh et al. [26] introduced a theoretical framework to guide the study and design of learnersourcing systems, combining four design questions common in crowdsourcing systems with two questions focused on contributors' skills and learning outcomes. They highlighted the importance of initially addressing the questions of "what is being done?" and "what are contributors learning from the task?" in any new projects.

One of the most widely cited examples of learnersourcing is the PeerWise system [27]. With PeerWise, students create multiple-choice questions (MCQs), and answer and review MCQs created by their peers. When authoring a new MCQ, a student provides the question text along with answer choices, marking which of the choices is correct. Research exploring the benefits of PeerWise has shown that authoring MCQs can measurably improve student learning [28, 29, 30]. Other related work has focused on improving student engagement, including exploring the use of various types of gamification, such as leaderboards, points and badges [31, 32].

A more recent learnersourcing system is RiPPLE [33], which is the platform we use in the current study. Similar to PeerWise, RiPPLE supports MCQs, but also includes support for other types of learning resources such as worked examples. Recent research involving the RiPPLE platform has explored the potential of learnersourcing as an alternative approach for evaluating the quality of learning resources, finding a strong correlation between student and expert ratings of artefacts while acknowledging variations in students' evaluation skills [34]. Research involving the RiPPLE platform has also examined approaches for modelling learners in learnersourcing contexts [35, 36], using AI-assisted methods of improving peer feedback [37, 38] and the use of explainable AI in education [39].

In the context of programming, a number of domain-specific learnersourcing systems have been developed. For example, both CodeWrite [40] and CrowdSorcerer [41] support having students create programming exercises, while SQL Trainer supports the creation of SQL exercises [42]. In addition to creating exercises, learnersourcing has been

used for other tasks. These include, for example, subgoal labels for how-to videos [43], personalized hints [44], and pedagogic explanations [45].

Khosravi et al. recently proposed a framework to guide the development of new learnersourcing approaches that leverage generative AI [46]. Their framework considers several key questions relating to human-AI partnerships that align with different aspects of learnersourcing, for example creating novel content, evaluating content quality, utilising learnersourced contributions and enabling instructors to provide support for students engaging in learnersourcing activities. In the current work, we focus on the 'Create' pillar of the framework proposed by Khosravi et al., by exploring the utility of learning resources generated by large language models.

2.4 Research Gap

As discussed in the previous two subsections, prior work has evaluated both learnersourced artefacts and artefacts generated by LLMs. However, to the best of our knowledge, no prior work has evaluated these at the same time, in the same context. Thus, there currently exists no empirical evidence of the differences in quality between student-created and LLM-created artefacts.

In the present work, we address this research gap by evaluating LLM-created and student-created artefacts in-situ as an authentic learning task, where the artefacts created by students and LLMs are mixed, i.e., the students who review the artefacts do so blind without knowing the source of each artefact. This approach is intended to give an accurate, authentic ranking of the quality of these artefacts.

3 Methods

In this section we describe how artefacts created by students participating in a learnersourcing activity were compared to those generated by an LLM. To begin, we provide an overview of the course context and the learnersourcing activity that was integrated into the course. Next, we introduce the research tool, *{AnonymizedTool}*, which was used to conduct the study. Lastly, we outline the study's design, detail the data collected, and discuss the measures used for analysis.

3.1 Course context

Our study was conducted in a large first-year programming course taught at the University of Auckland. This course is compulsory for all first-year students in the engineering programme, uses C as the language of instruction, and had an enrollment of 950 students in 2022. The course is competitive as, along with all other first-year courses, performance is used to rank students in order to determine entry into limited-place specialisations in the second year. Typically, fewer than 5% of enrolled students withdraw from or fail the course.

3.2 Learnersourcing task

My code:

```
int SumBetween(int low, int high)
{
    int sum = 0;
    int i;
    for (i = low; i <= high; i++) {
        sum = sum + i;
    }
    return sum;
}
```

My explanation: This function calculates the sum of all values between the inputs low and high. It declares a variable to store the sum, and sets this to zero. It then uses a 'for' loop to iterate over all values between low and high (inclusive). For each value, the value is added to the sum variable. When the loop finishes, the value of sum is returned.

Figure 1: A sample "code example" artefact

The learnersourcing task took place during one of the weekly lab sessions. The lab session is a graded component of the course, and completion contributes 2% towards each student's grade. The lab consisted of four short programming tasks, which were completed individually, and the learnersourcing activity was the final task of the lab. For this task, students were asked to produce a "code example" which was defined to consist of two parts, as shown in Figure 1: (1) a "My code" section, where the student had the freedom to write a fragment of code of any length and complexity. The instructions stated: "You can provide any code you like for your example. This could be a function you have written, an entire program or even just a short fragment consisting of a few lines of code. It is completely up to you. Ideally, the code you write should highlight one or more concepts that are important in the context of the course. Avoid targeting concepts that are not relevant to our course."; and (2) a "My explanation" section, where the student was required to provide a natural language explanation for the code fragment. Again, students were mostly free to choose the level of detail contained in their explanation. At a minimum, they were expected to provide a high-level description of the code. The following requirements were stated: "The explanation is simply a short paragraph of text. You should start your explanation with one sentence that describes the overall purpose of the code. After this sentence, you can provide additional details, such as explaining the purpose or function of specific lines of code. Overall, your explanation does not need to be more than a few sentences in length (although feel free to write more)".

3.3 Research tool:*{AnonymizedTool}*

For our study, we made use of *{AnonymizedTool}* [33], which is an adaptive educational system that develops learning content through learnersourcing. This system offers three types of learning activities: *create*, *evaluate*, and *practice*. We now briefly describe each of these activities.

Create. Students in a course develop study resources like MCQs and worked examples. To conduct our study, we introduced a new resource type, "code example", as previously described, to the system. To incorporate our new resource type, "code example," we prompted students to select relevant concepts from a list of course topics when creating a new example. They were also asked to indicate the example's level of difficulty (easy, medium, or hard) and provide a brief title. Figure 2 depicts the interface presented to students for creating a "code example" in *{AnonymizedTool}*.

Evaluate. The study resources created by students undergo peer review by multiple peers. Reviewers use a set of criteria to rate the quality of the resource and their confidence in their rating. Additionally, they provide written feedback that justifies their rating and provides constructive feedback on how to improve the resource. Figure 3 displays the interface and evaluation criteria for assessing the quality of a code example. Using the submitted peer reviews and algorithms discussed in [37], *{AnonymizedTool}* makes a determination on the quality of the resource. Approved resources are added to a course repository that is accessible to all students while ineffective resources are returned to the author with feedback for resubmission. Instructors can monitor the process with the help of an AI-based spot-checking algorithm [25].

Practice. The approved resource repository is utilized to provide personalized practice resources to individual students based on their current learning needs.

3.4 Experimental Design

To address our research questions, we conducted a study using *{AnonymizedTool}* in which students created code examples as part of one activity in a weekly lab session. Students were provided with six exemplars created by the teaching team to help them craft their own code examples. The code for all six exemplars was provided in the form of a single function definition. Table 1 shows all six exemplars (code is not included for space reasons). Three of these six examples (i.e. PrintAverageRainfall, SumBetween and PrintSummary) were published explicitly as part of the description of the lab activity, and the other three were made available as part of a lecture that accompanied the release of the lab resources.

In a separate process, we used these same six examples to prompt Codex (code-davinci-002; an OpenAI model optimized for code-completion tasks) to generate code examples and their explanations. We used the default temperature of 0.7 to generate the code examples. In total, we generated 100 novel code examples using the Codex model. We began by constructing a prompt with a structure similar to that shown in Listing 1. The keyword "EXAMPLE" was used to denote the beginning of a new example, and each example was divided into a "Code" and "Explanation" subsection. Listing 1 shows two of the six exemplars, but does not include the other exemplars or the full code and explanations for space reasons. Of the 100 code examples we generated, 50 were produced using a shorter prompt consisting of a subset of the three exemplars (the three that were published in the description for the lab activity), and the remaining 50 were produced with a longer prompt that included all six exemplars (the three published in the lab description and the three in the accompanying lecture). The purpose of this split was to facilitate exploration into whether the number of few-shot examples given to the model would measurably affect the quality of the AI-generated resources. This analysis

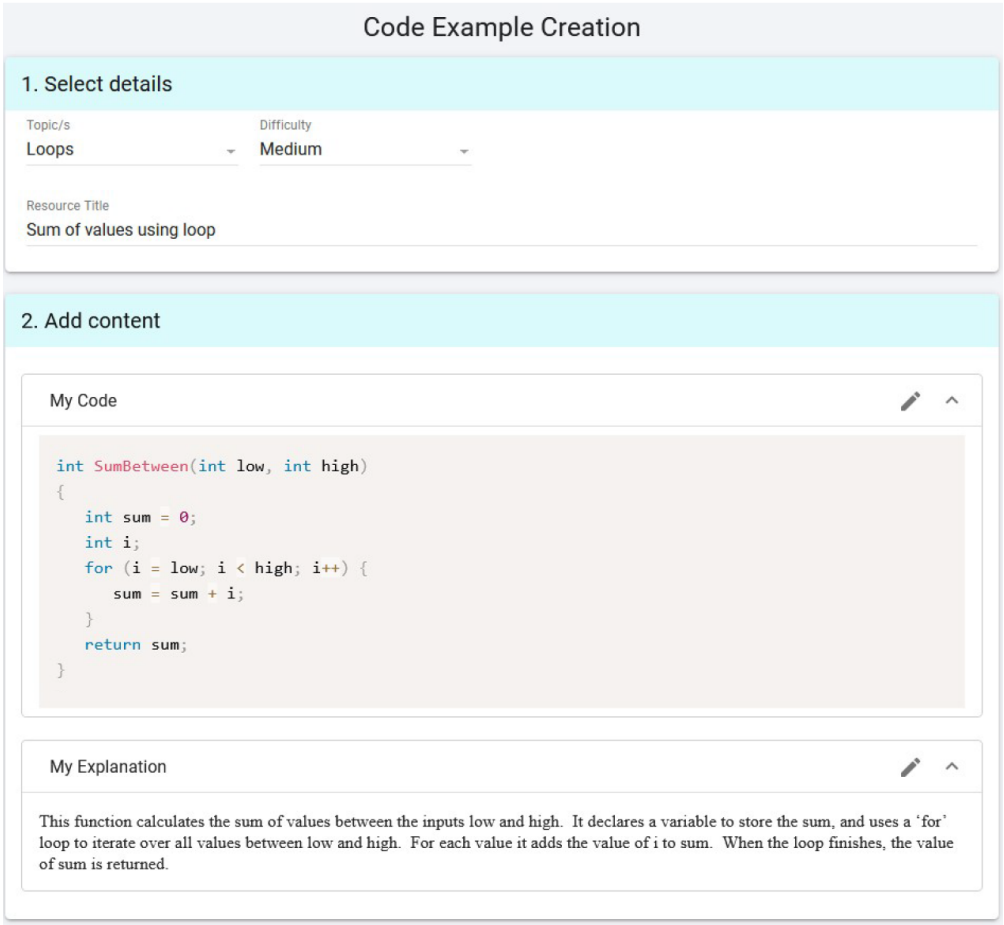

Figure 2: The user interface of *{AnonymizedTool}* showing the creation process for a new "code example" artefact.

is outside the scope of the current study and left for future work. The prompt ended with the keyword "EXAMPLE" followed by the keyword "Code:" which was intended to prompt the completion model to generate a new example.

Figure 4 illustrates two of the new code examples that were generated by the Codex model. All 100 code examples generated by Codex were then inserted into *{AnonymizedTool}* and appeared in the same pool alongside the student-generated resources.

Subsequently, as part of the lab activity, all of the code examples that were produced by students and generated by Codex were peer-reviewed by students using *{AnonymizedTool}*'s evaluation functionality. Notably, the source of the resources was not shown to students during this review phase, thus the peer evaluation was performed blind. Table 2 displays the count of the number of resources that were produced by students and by Codex, along with the total number of peer reviews submitted for these resources.

3.5 Metrics and Analysis.

In this section, we describe the metrics employed and analyses conducted to address our research questions. We employed Mann-Whitney U tests to analyze quantitative data and the Chi-squared test of association for categorical data. A significance criterion of $p < 0.05$ was adopted for assessing statistical significance.

RQ1: Comparison of Resources. Our aim for RQ1 is to assess the comprehensiveness of the resources generated by students and by the AI model. We explore the reserved C keywords used in the "code" component of the resources created by Codex, students and instructors (i.e. the six exemplars). Additionally, we gauge the length (character count) of both the "code" and "explanation" components as an indicator of their comprehensiveness. We present the outcomes of this investigation in Section 4.1.

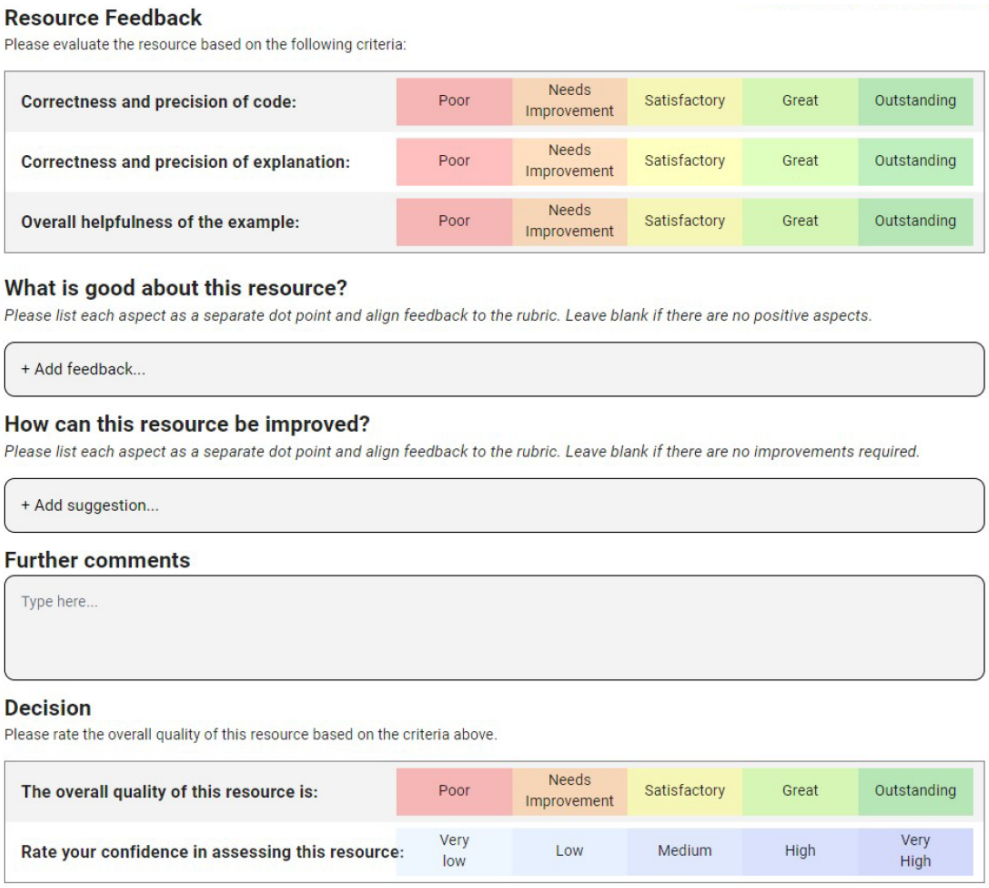

Figure 3: The evaluation interface used for reviewing the quality of a "code example"

```
int BiggestInRange(int *values, int start, int end)
{
    int max = values[start];
    int i;
    for (i = start + 1; i <= end; i++) {
        if (values[i] > max) {
            max = values[i];
        }
    }
    return max;
}
```

This function finds the largest value in a segment of an array. It starts by setting the maximum value to the value at the starting index position. Then it loops over the values between the starting and ending index positions, and updates the maximum value as required. When the loop finishes, the maximum value is returned.

```
void RemoveChar(char *string, char c) {
    int i, j;
    for (i = 0; i < strlen(string); i++) {
        if (string[i] == c) {
            for (j = i; j < strlen(string); j++) {
                string[j] = string[j+1];
            }
        }
    }
}
```

This code removes all occurrences of character c in string. It starts by iterating over all characters in the string. For each character, it checks to see if the character is equal to the required character. If it is, then the code shifts all characters from this point onwards one position to the left. This has the effect of removing the character from the string.

Figure 4: Two new "code examples" generated as output by Codex

RQ2: Perceived Quality. For RQ2, we examined students' perceptions of the quality of student and AI-generated content by analysing their responses to each criteria (code correctness, explanation correctness, and example helpfulness) and their overall decisions regarding the quality of the resources. We present the findings of this investigation in Section 4.2.

4 Results

In this section we report the results of our investigation into answering the two research questions proposed in Section 1.

Table 1: Complete list of exemplars that were provided to students and used to formulate the prompt given to the Codex model (in the format shown in Listing 1). Code is not shown for space reasons.

Function header	Explanation
void PrintAverageRainfall(int values[])	This code is used to get the average rainfall by filtering for the non-negative numbers of the array and adding the numbers up together. Then it calculates the average by dividing the sum by the count of the non-negative numbers. The corresponding result is printed out if the array contains more than one values, and "no rain" is printed out otherwise.
int SumBetween(int low, int high)	This function calculates the sum of all values between the inputs low and high. It declares a variable to store the sum, and sets this to zero. It then uses a 'for' loop to iterate over all values between low and high (inclusive). For each value, the value is added to the sum variable. When the loop finishes, the value of sum is returned.
void PrintSummary(int values[])	The purpose of this code is to print out whether there are more positive numbers or negative numbers in an array. First, variables are declared for counting the number of positive and negative values in the array. Then, there is a loop to iterate through the array. Each time this loop goes through, it will check if the value is positive or negative and count that specific value appropriately. After all numbers are counted correctly, the loop will check if there are more positive or more negative numbers and print out the correct response.
int IsSuffix(char *suffix, char *word)	This function tests whether one string ends with a complete copy of another string, forming a suffix. It starts by calculating the length of both strings. Then it iterates over each character in the potential suffix, comparing it to the correpsonding character in the original string such that the lengths line up. If a character is found not to match, then the function returns 0. If the loop finishes, then it means all characters match so the function returns 1.
void RemoveLetterAt(char *word, int position)	This function removes the character at the specified index position in the string. It starts looping over the characters in the string beginning from the specified index position, shifting each character one position to the left. This eventually copies the null character one position to the left, shortening the string as required.
void ReverseArray(int *values, int left, int right)	This function reverses all of the elements in an array that lie between index positions left and right inclusive. It starts by calculating the length of the segment of the array that lies between the left and right index positions. It then loops over the first half of the values in this segment, swapping each value with its corresponding value at the other end of the segment. To swap the values, a temporary variable is used. On each iteration, the left value increases and the right value decreases.

Table 2: Number of resources created by students and by Codex, and the number of peer evaluations received

Resource type	# Resources	# Evaluation
Student-generated	886	4,053
AI-generated	101	446
All	987	4,499

4.1 RQ1: Comprehensiveness of resources created by students vs AI

Tables 3, 4, and 5 compare the occurrences for reserved C keywords in the instructor, AI-generated, and student-created code examples. Only keywords that had occurrences for at least two categories (instructor, AI, student) are presented in the tables. In addition to the values shown in the tables, only students had some occurrences for the following keywords: 'float', 'do', 'case', 'continue', 'short', 'unsigned', 'true', 'struct', 'const', 'false', 'default', 'typedef', 'auto', 'goto', 'signed', 'static', 'switch'.

Table 3 shows the raw number of occurrences for each keyword in the instructor, the AI-generated, and the student-created code examples. Table 4 show the average number of times the keyword appeared in a single code example, while Table 5 shows the average number of occurrences per 1000 characters of code. The values in Table 4 can be thought of as being normalized with regards to the *number of resources created* while the values in Table 5 can be thought of as being normalized with regards to *both the number and the length* of the resources created.

Listing 1: Prompt structure for generating Codex code examples and explanations

```
EXAMPLE:

Code:
void PrintAverageRainfall(int values[])
{
    ...
}

Explanation:
This code is used...

EXAMPLE:

Code:
int SumBetween(int low, int high)
{
    ...
}

Explanation:
This function calculates the ...

...

EXAMPLE:

Code:
```

There are some interesting observations with respect to Table 5. All three groups – instructors, AI, students – have different behaviors with regards to keyword use. One interesting observation is that code examples produced by the AI very rarely use the keyword 'double' compared to examples produced by instructors and students, with students using 'double' the most. The AI uses 'while' frequently compared to humans: four times more than students, and slightly under twice as often as instructors. The AI and the instructors use 'while' and 'for' approximately as often, while students use 'while' a lot less often than 'for', using 'for' three times more often than 'while'. Both the AI and students use 'else' much less frequently than the instructors. The AI uses 'return' almost twice as often as the humans. Students use 'if' slightly less compared to instructors and the AI. Interestingly, the number of occurrences of the 'int' keyword is different for all three groups: students use it the least, instructors use it more than students, but the AI uses it the most. The difference in the distributions between all three groups (based on the raw number of keyword occurrences) is statistically significant using a Chi-squared test ($p < .001$).

Table 3: Comparison of instructor vs AI vs student-generated content based on the raw occurrences of the keywords

group	int	if	return	void	else	for	while	char	double	sizeof	break	long
Instructor	24	7	4	4	4	3	3	3	1	0	0	0
AI	378	83	90	40	13	56	54	31	1	5	2	2
Students	3765	1442	921	706	431	1000	319	650	366	13	45	8

Table 4: Comparison of instructor vs AI vs student-generated content based on how many times that keyword occurs in a created resource on average

group	int	if	return	void	else	for	while	char	double	sizeof	break	long
Instructor	4.0	1.17	0.67	0.67	0.67	0.5	0.5	0.5	0.17	0.0	0.0	0.0
AI	3.74	0.82	0.89	0.4	0.13	0.55	0.53	0.31	0.01	0.05	0.02	0.02
Students	4.25	1.63	1.04	0.8	0.49	1.13	0.36	0.73	0.41	0.01	0.05	0.01

Table 6 presents the lengths of both the code and explanation components of the code examples created by students and the AI. From the table, we can observe that there is a considerable (and statistically significant, Mann-Whitney U = 25123.5, p < .001) difference in the lengths of the code components produced, with the median length being 224

Table 5: Comparison of instructor vs AI vs student-generated content based on how often the keyword appears in 1000 characters long code on average

group	int	if	return	void	else	for	while	char	double	sizeof	break	long
Instructor	10.76	3.14	1.79	1.79	1.79	1.34	1.34	1.34	0.45	0.0	0.0	0.0
AI	14.86	3.26	3.54	1.57	0.51	2.2	2.12	1.22	0.04	0.2	0.08	0.08
Students	6.13	2.35	1.5	1.15	0.7	1.63	0.52	1.06	0.6	0.02	0.07	0.01

characters for the AI and 377 for the students. For the explanations, the difference is not as large although it is still statistically significant (U = 55834.5, p < .001), and the direction is the opposite, with students producing a median of 303 characters for their explanations compared to 395 characters for the AI.

Table 6: Lengths of codes and explanations created by students and AI in characters (rounded to nearest integer).

	Code			Explanation		
	Mdn	μ	σ	Mdn	μ	σ
Students	377	693	1750	303	373	329
AI	224	252	113	395	405	133

4.2 RQ2: Student perceptions of the quality of the resources created by students vs AI

Table 7: Comparison of student vs AI-generated content based on code correctness, explanation correctness, and helpfulness of the generated resource.

	Code correctness			Explanation correctness			Helpfulness		
	Mdn	μ	σ	Mdn	μ	σ	Mdn	μ	σ
Students	4	4.09	0.91	4	4.01	0.93	4	3.91	0.94
AI	4	4.03	0.98	4	4.10	0.87	4	3.89	0.97

Table 7 presents student perceptions on the quality of the resources related to correctness of both the code and the explanation and the perceived helpfulness of the resource. From the table, we can observe that while the medians are the same (4 for all evaluated aspects), there are slight differences in the means, although none of the differences are statistically significant (U = 885785.0, p = 0.46 for code correctness; U = 940809.5, p = 0.13 for explanation correctness; U = 894970.0, p = 0.72 for helpfulness).

Table 8 presents student perceptions on the overall quality of the resources, as indicated by their responses to the 'decision' field on the evaluation form (this appears towards the bottom of the form as shown in Figure 3. This 'decision' criteria measures the overall quality of the resource, and the confidence of the rater in their decision. From the table we can see that while the medians are the same (4 across the board), there are again slight differences in the means. However, the differences are not statistically significant (U = 860373.5, p = 0.07 for decision; U = 922883.5, p = 0.43 for confidence).

5 Discussion

Results of RQ1 suggest that student-generated learning resources exhibit noticeable variety regarding their overall length and syntax features. In terms of syntax features, the AI, instructors, and students all differ in their usage of specific programming keywords. The divergent use of certain keywords between the AI and students suggests that the AI models may prioritize different programming constructs such as 'while' and 'return'. This could have both potential advantages and disadvantages. On the one hand, it can potentially limit the variety of code examples learners are exposed to, impacting their understanding of alternative coding strategies. However, it could also be beneficial if those constructs are indeed more important with respect to the course or applicable to a wider range of problems. Further investigation is needed to determine the impact of this difference on learning outcomes.

The length of AI-generated code appears to be comparable to that of instructors', generally offering more concise functions which might prove more efficient or easier for novices to comprehend. Several factors contribute to the more extended and varied length observed in student-generated content. It seems that students embrace a broader interpretation of the "code example" resource type, rather than more closely adhering to the kinds of examples provided by the instructors. Some students included various components of a full-fledged program, such as include statements, test cases, and comments, in contrast to the instructors' examples which comprised a single function with no comments.

Table 8: Comparison of student vs AI-generated content based on the overall 'decision' (overall quality) and confidence in the rater's decision.

	Decision			Confidence		
	Mdn	μ	σ	Mdn	μ	σ
Students	4	3.94	0.91	4	3.95	0.78
AI	4	3.85	0.97	4	3.97	0.81

These findings point to a greater level of autonomy in students' code writing style, while the AI seems to mirror the given examples and instructions more closely. Explanations generated by the AI are typically similar in length to the instructor's examples and capitalize on the capability of large language models for producing high-quality writing. We observed almost no grammatical or spelling errors in the explanations produced by Codex. The longer explanations produced by the AI, compared to the student explanations, can be seen as a benefit as it may provide more in-depth coverage of the code. In contrast, student explanations tend to be significantly shorter but exhibit a larger variation in length. This could be attributed to the diverse backgrounds of the student population, which might encompass substantial differences in English language proficiency.

The results of RQ2 demonstrate a comparable perceived quality between student-generated and AI-generated learning resources, in terms of both correctness and helpfulness. This outcome suggests that AI-generated resources could be a viable supplement or alternative to student-generated resources in certain learning contexts. A slightly higher average rating for student-generated code correctness might indicate that students appreciate the broader variety in code examples that may include test cases and comments. On the other hand, the slightly higher mean rating for AI-generated explanations could suggest that the AI's ability to provide detailed and coherent insights is recognized and appreciated by students. Nevertheless, students appear to view student-generated and AI-generated resources largely in similar ways. However, given the close ratings, further research could be beneficial to understand under what conditions one type of resource might be preferred over the other. For example, in our study the explanations that are listed in Table 1 appear quite homogeneous. Incorporating a wider variety in terms of length and difficulty in the exemplar resources that are provided as prompts to the AI model could potentially stimulate the creation of more diverse learning materials. Such an expanded set of examples can provide a richer experience for students and more diverse learning resources. Also, there is a need for studies to examine the long-term impact of the use of AI-generated resources on learning outcomes and to ascertain whether these initial perceptions of the usefulness of AI-generated content hold up over time.

This initial study highlights several potentially fruitful avenues for future work. As mentioned above, although we did not observe statistically significant differences in the perceived quality of AI-generated and student-generated resources, such difference might become apparent in larger populations of within different learning contexts. For example, the student cohort in our study are highly incentivised to perform well, and thus are typically well engaged with learning activities. In situations where students are less engaged, learnersourcing activities are often less successful due to the low quality of student generated content. In such settings, resources that are automatically generated by AI models and which align well to instructor resources may be particularly valuable. The rapid and continuous improvement in LLMs also presents exciting opportunities for the future of AI-generated learning resources. For example, it may be worthwhile to investigate whether LLMs can be prompted to generate explanations that cater for specific knowledge levels, such as simplifying concepts for students with less strong backgrounds and providing advanced instruction for more capable students.

6 Conclusion

In this paper, we explore the potential of employing large language models (LLMs) for generating learning resources in the context of an introductory programming course by contrasting them with resources created by students. Using a blind evaluation we examine the perceived correctness and helpfulness of both AI-generated and student-generated resources.

Our primary findings reveal that the perceived quality of AI-generated resources is largely on par with student-generated resources, though there are notable differences in terms of syntax features and length. This suggests that AI-generated content may serve as a viable supplement or as an alternative to traditional learnersourcing techniques in certain learning contexts. However, the limited variety in AI-generated examples and their close adherence to given prompts raises questions about their adaptability to cater to diverse learning needs and preferences.

In terms of broader lessons, this research highlights the potential for LLMs to improve the accessibility and scalability of high-quality educational content, while reducing the burden on educators. Future work should focus on investigating the limitations identified in this study, including using a wider variety of input prompts, and seeking to understand

the long-term impact of AI-generated resources on learning outcomes. Our evaluation was also limited to one type of learning resource suitable for introductory programming courses, and future work should explore different kinds of resources in a broad range of subject areas. The automatic generation of learning resources using LLMs holds great promise for enriching the educational landscape across various domains and skill levels.

Statements on open data, ethics and conflict of interest

- **Open data**: All data that was analysed for this work will be made publicly available, along with corresponding Jupyter notebook scripts.
- **Ethics statement**: This research was conducted with approval of the {*Anonymized - details present on title page*}
- **Conflict of interest statement**: There are no conflicts of interest to declare.

References

[1] Ricardo Torres Kompen, Palitha Edirisingha, Xavier Canaleta, Maria Alsina, and Josep Maria Monguet. Personal learning environments based on web 2.0 services in higher education. *Telematics and Informatics*, 38:194–206, 2019. ISSN 0736-5853. doi:https://doi.org/10.1016/j.tele.2018.10.003. URL `https://www.sciencedirect.com/science/article/pii/S0736585318306312`.

[2] Juho Kim. *Learnersourcing: improving learning with collective learner activity*. PhD thesis, Massachusetts Institute of Technology, 2015.

[3] Nea Pirttinen, Paul Denny, Arto Hellas, and Juho Leinonen. Lessons learned from four computing education crowdsourcing systems. *IEEE Access*, 11:22982–22992, 2023. doi:10.1109/ACCESS.2023.3253642.

[4] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 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*, ICER '22, page 27–43, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391948. doi:10.1145/3501385.3543957. URL `https://doi.org/10.1145/3501385.3543957`.

[5] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. Comparing code explanations created by students and large language models. In *Proceedings of the 28th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, ITiCSE '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701382. doi:10.1145/3587102.3588785. URL `https://doi.org/10.1145/3587102.3588785`.

[6] Zichao Wang, Jakob Valdez, Debshila Basu Mallick, and Richard G Baraniuk. Towards human-like educational question generation with large language models. In *Artificial Intelligence in Education: 23rd International Conference, AIED 2022, Durham, UK, July 27–31, 2022, Proceedings, Part I*, pages 153–166. Springer, 2022.

[7] Sahan Bulathwela, Hamze Muse, and Emine Yilmaz. Scalable educational question generation with pre-trained language models, 2023.

[8] Emilio Ferrara. Should chatgpt be biased? challenges and risks of bias in large language models. *arXiv preprint arXiv:2304.03738*, 2023.

[9] Mike Perkins. Academic integrity considerations of ai large language models in the post-pandemic era: Chatgpt and beyond. *Journal of University Teaching & Learning Practice*, 20(2):07, 2023.

[10] Tamara Tate, Shayan Doroudi, Daniel Ritchie, Ying Xu, et al. Educational research and ai-generated writing: Confronting the coming tsunami, 2023.

[11] Lixiang Yan, Lele Sha, Linxuan Zhao, Yuheng Li, Roberto Martinez-Maldonado, Guanliang Chen, Xinyu Li, Yueqiao Jin, and Dragan Gašević. Practical and ethical challenges of large language models in education: A systematic literature review. *arXiv preprint arXiv:2303.13379*, 2023.

[12] Thomas K.F. Chiu, Qi Xia, Xinyan Zhou, Ching Sing Chai, and Miaoting Cheng. Systematic literature review on opportunities, challenges, and future research recommendations of artificial intelligence in education. *Computers and Education: Artificial Intelligence*, 4:100118, 2023. ISSN 2666-920X. doi:https://doi.org/10.1016/j.caeai.2022.100118. URL `https://www.sciencedirect.com/science/article/pii/S2666920X2200073X`.

[13] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 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*, SIGCSE 2023, page 500–506, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394314. doi:10.1145/3545945.3569759. URL https://doi.org/10.1145/3545945.3569759.

[14] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stepha Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023. ISSN 1041-6080. doi:https://doi.org/10.1016/j.lindif.2023.102274. URL https://www.sciencedirect.com/science/article/pii/S1041608023000195.

[15] 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. Computing education in the era of generative ai. *arXiv preprint*, 2023. doi:10.48550/arXiv.2306.02608.

[16] Arun Raman and Viraj Kumar. Programming pedagogy and assessment in the era of ai/ml: A position paper. In *Proceedings of the 15th Annual ACM India Compute Conference*, COMPUTE '22, page 29–34, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450397759. doi:10.1145/3561833.3561843. URL https://doi.org/10.1145/3561833.3561843.

[17] Michel Wermelinger. Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 172–178, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394314. doi:10.1145/3545945.3569830. URL https://doi.org/10.1145/3545945.3569830.

[18] Jaromir Savelka, Arav Agarwal, Christopher Bogart, Yifan Song, and Majd Sakr. Can generative pre-trained transformers (gpt) pass assessments in higher education programming courses? *arXiv preprint*, 2023. doi:10.48550/arXiv.2303.09325.

[19] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, page 10–19, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450396431. doi:10.1145/3511861.3511863. URL https://doi.org/10.1145/3511861.3511863.

[20] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 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*, ACE '23, page 97–104, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399418. doi:10.1145/3576123.3576134. URL https://doi.org/10.1145/3576123.3576134.

[21] Paul Denny, Viraj Kumar, and Nasser Giacaman. 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 - Volume 1*, SIGCSE 2023, page 1–7, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394314. doi:10.1145/3545945.3569823. URL https://doi.org/10.1145/3545945.3569823.

[22] Stephen R. Piccolo, Paul Denny, Andrew Luxton-Reilly, Samuel Payne, and Perry G. Ridge. Many bioinformatics programming tasks can be automated with chatgpt. *arXiv preprint*, 2023. doi:10.48550/arXiv.2303.13528.

[23] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proc. SIGCSE'23*. ACM, 2023.

[24] Paul Denny, Diana Cukierman, and Jonathan Bhaskar. Measuring the effect of inventing practice exercises on learning in an introductory programming course. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, page 13–22, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450340205. doi:10.1145/2828959.2828967. URL https://doi.org/10.1145/2828959.2828967.

[25] Hassan Khosravi, Gianluca Demartini, Shazia Sadiq, and Dragan Gasevic. Charting the design and analytics agenda of learnersourcing systems. In *LAK21: 11th International Learning Analytics and Knowledge Conference*, LAK21, page 32–42, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450389358. doi:10.1145/3448139.3448143. URL https://doi.org/10.1145/3448139.3448143.

[26] Anjali Singh, Christopher Brooks, and Shayan Doroudi. Learnersourcing in theory and practice: Synthesizing the literature and charting the future. In *Proceedings of the Ninth ACM Conference on Learning @ Scale*, L@S '22, page 234–245, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391580. doi:10.1145/3491140.3528277. URL https://doi.org/10.1145/3491140.3528277.

[27] Paul Denny, John Hamer, Andrew Luxton-Reilly, and Helen Purchase. Peerwise: students sharing their multiple choice questions. In *Proceedings of the fourth international workshop on computing education research*, pages 51–58, 2008.

[28] Piers DL Howe, Meredith McKague, Jason M Lodge, Anthea G Blunden, and Geoffrey Saw. Peerwise: Evaluating the effectiveness of a web-based learning aid in a second-year psychology subject. *Psychology Learning & Teaching*, 17(2):166–176, 2018.

[29] Paul Denny, Brian Hanks, and Beth Simon. Peerwise: replication study of a student-collaborative self-testing web service in a us setting. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 421–425, 2010.

[30] Matthew R. Kelley, Elizabeth K. Chapman-Orr, Susanna Calkins, and Robert J. Lemke. Generation and retrieval practice effects in the classroom using peerwise. *Teaching of Psychology*, 46(2):121–126, 2019. doi:10.1177/0098628319834174. URL https://doi.org/10.1177/0098628319834174.

[31] Paul Denny, Fiona McDonald, Ruth Empson, Philip Kelly, and Andrew Petersen. Empirical support for a causal relationship between gamification and learning outcomes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi:10.1145/3173574.3173885. URL https://doi.org/10.1145/3173574.3173885.

[32] Mitchell Rogers, Wendy Yao, Andrew Luxton-Reilly, Juho Leinonen, Danielle Lottridge, and Paul Denny. Exploring personalization of gamification in an introductory programming course. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 1121–1127, 2021.

[33] Hassan Khosravi, Kirsty Kitto, and Joseph Jay Williams. RiPPLE: A crowdsourced adaptive platform for recommendation of learning activities. *Journal of Learning Analytics*, 6(3):91–105, 2019.

[34] Solmaz Abdi, Hassan Khosravi, Shazia Sadiq, and Gianluca Demartini. Evaluating the quality of learning resources: A learnersourcing approach. *IEEE Transactions on Learning Technologies*, 14(1):81–92, 2021. doi:10.1109/TLT.2021.3058644.

[35] Hatim Lahza, Hassan Khosravi, and Gianluca Demartini. Analytics of learning tactics and strategies in an online learnersourcing environment. *Journal of Computer Assisted Learning*, 39(1):94–112, 2023.

[36] Solmaz Abdi, Hassan Khosravi, Shazia Sadiq, and Ali Darvishi. Open learner models for multi-activity educational systems. In *Artificial Intelligence in Education: 22nd International Conference*, pages 11–17. Springer, 2021.

[37] Ali Darvishi, Hassan Khosravi, Solmaz Abdi, Shazia Sadiq, and Dragan Gašević. Incorporating training, self-monitoring and ai-assistance to improve peer feedback quality. In *Proceedings of the Ninth ACM Conference on Learning@ Scale*, pages 35–47, 2022.

[38] Ali Darvishi, Hassan Khosravi, Shazia Sadiq, and Dragan Gašević. Incorporating ai and learning analytics to build trustworthy peer assessment systems. *British Journal of Educational Technology*, 53(4):844–875, 2022.

[39] Hassan Khosravi, Simon Buckingham Shum, Guanliang Chen, Cristina Conati, Yi-Shan Tsai, Judy Kay, Simon Knight, Roberto Martinez-Maldonado, Shazia Sadiq, and Dragan Gašević. Explainable artificial intelligence in education. *Computers and Education: Artificial Intelligence*, 3:100074, 2022.

[40] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. Codewrite: supporting student-driven practice of java. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 471–476, 2011.

[41] Nea Pirttinen, Vilma Kangas, Irene Nikkarinen, Henrik Nygren, Juho Leinonen, and Arto Hellas. Crowdsourcing programming assignments with crowdsorcerer. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 326–331, 2018.

[42] Juho Leinonen, Nea Pirttinen, and Arto Hellas. Crowdsourcing content creation for sql practice. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 349–355, 2020.

[43] Sarah Weir, Juho Kim, Krzysztof Z Gajos, and Robert C Miller. Learnersourcing subgoal labels for how-to videos. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, pages 405–416, 2015.

[44] Elena L Glassman, Aaron Lin, Carrie J Cai, and Robert C Miller. Learnersourcing personalized hints. In *Proceedings of the 19th ACM conference on computer-supported cooperative work & social computing*, pages 1626–1636, 2016.

[45] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z Gajos, Walter S Lasecki, and Neil Heffernan. Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, pages 379–388, 2016.

[46] Hassan Khosravi, Paul Denny, Steven Moore, and John Stamper. Learnersourcing in the age of ai: Student, educator and machine partnerships for content creation. *arXiv preprint*, 2023. doi:10.48550/arXiv.2306.06386.