Engaging an LLM to Explain Worked Examples for Java Programming: Prompt Engineering and a Feasibility Study

Mohammad Hassany[1], Peter Brusilovsky[1], Jiaze Ke[2], Kamil Akhuseyinoglu[1] and Arun Balajiee Lekshmi Narayanan[1]

[1]University of Pittsburgh, Pittsburgh, PA
[2]Carnegie Mellon University, Pittsburgh, PA

Abstract

Worked code examples are among the most popular types of learning content in programming classes. Most approaches and tools for presenting these examples to students are based on line-by-line explanations of the example code. However, instructors rarely have time to provide line-by-line explanations of a large number of examples typically used in a programming class. This paper explores the opportunity to facilitate the development of worked examples for Java programming through a human-AI collaborative authoring approach. The idea of collaborative authoring is to generate a starting version of code explanations using LLM and present it to the instructor to edit if necessary. The critical step towards implementing this idea is to ensure that LLM can produce code explanations that look meaningful and acceptable to instructors and students. To achieve this goal, we performed an extensive prompt engineering study and evaluated the explanation produced by the selected prompt in a user study with students and authors.

Keywords

Code Examples, Authoring Tool, Human-AI Collaboration

1. Introduction

Program code examples play a crucial role in learning to program [1]. Instructors use examples extensively to demonstrate the semantics of the programming language being taught and to highlight the fundamental coding patterns. Programming textbooks also pay a lot of attention to examples, with a considerable textbook space allocated to program examples and associated comments.

Through this practice, worked code examples emerged as an important type of learning content in programming classes. Following the tradition established by a number of programming textbooks [2, 3], a typical worked example presents a code to solve a specific programming problem and explains the role and function of code lines or code chunks. In textbooks, these explanations are usually presented as comments in the code or as explanations in the margins. Although informative, this approach focused on passive learning, which is known for its low efficiency. Recognizing this problem, several research teams developed learning tools that offered more interactive and engaging ways to learn from examples [4, 5, 6, 7, 8]. These tools demonstrated their effectiveness in classroom studies, but their practical impact, i.e., wider use by programming instructors, was limited due to *authoring bottleneck*. Although the authors of example-focused learning tools usually provide a good set of worked examples that can be presented through their tools, many instructors prefer to use their own favorite code examples. Instructors are usually happy to broadly share the code of examples they created (usually providing it on the course Web page), but they rarely have time or patience to augment examples with explanations and add their examples to an example-focused interactive system. In fact, producing a single explained example could take 30 minutes or more, since it requires typing an explanation for each code line [4, 8] or creating a screencast in a specific format [5, 7].

The authoring bottleneck has been recognized by several research teams, which have offered several ways to address it. Among the approaches explored are learner-sourcing, that is, engaging students in creating and reviewing explanations for instructor-provided code [9] and automatic extraction of information content from available sources, such as lecture recordings [6]. In this paper, we explore an alternative approach to reduce the authoring bottleneck based on the human-AI collaborative authoring process. With this approach, the instructor provides the code of one of their favorite examples along with the statement of the programming problem that it is solving. The AI engine based on large language models (LLM) examines the code and generates explanations for each code line at several levels of detail. As an option, the instructor could edit and refine the text produced by LLM to adapt it to the goals of the class and the target students. As in any productive collaboration, each side does what it is best suited to do, leaving the challenging work to the partner. To support and explore this authoring approach, we are developing an authoring system with the aim to radically decrease the time to create a new interactive worked example. The examples created by the system could be uploaded to an example exploration system such as WebEx [4] or PCEX [8] or exported in a reusable format.

By its nature, our project itself is a collaboration between experts in HCI who focus on developing a usable human-AI collaboration interface through iterative design and evaluation and AI experts who focus on producing good code explanations with LLM through iterative prompt engineering and prompt evaluation. Our current work on developing and evaluating the human-AI authoring interface is presented in [10]. In this paper, we focus on generating usable code explanations with a popular LLM ChatGPT.

The remainder of the paper is structured as follows. We start by reviewing related work (Section 2), focusing on worked examples, as a special kind of learning content, and the current work on generating code explanations using ChatGPT. Next (Section 3), we review our prompt engineering experiments, in which we attempted to generate the best-performing prompt through an internal evaluation process. Among the options explored in this process, we assessed the value of using the problem statement to generate good explanations and minimize "hallucinations". Following that

EDM 2024 Workshop: Leveraging Large Language Models for Next Generation Educational Technologies,July 14, 2024, Atlanta, GA

(Section 4) we present a user study, which we performed to assess the quality of the resulting LLM explanations. In this study, teaching assistants (TAs) and students compared the code explanations created by experts through a traditional process with examples created by ChatGPT to contribute to human-AI collaborative authoring process. In Section 5 we conclude with a summary of the work and plans for future research.

2. Related Work

2.1. Worked Examples in Programming

Code examples are important pedagogical tools for learning programming. Not surprisingly, considerable effort has been devoted to the development of learning materials and tools to support students in studying code examples. Hosseini [8] classified the program examples that have been used in teaching and learning to program into two groups, according to their primary instructional goal: *program behavior examples* and *program construction examples*. Program behavior examples are used to demonstrate the semantics (i.e., behavior) of various programming constructs (i.e., what is happening inside a program or an algorithm when it is executed). Program construction examples attempt to communicate important programming patterns and practices by demonstrating the construction of a program that achieves various meaningful purposes.

Program behavior examples have been extensively studied. While textbooks still explain program behavior by using textual comments attached to lines of program code, a more advanced method for this purpose — *program visualization*, which visually illustrates the runtime behavior of computer programs — is now considered as state-of-the-art. During the past three decades, several specialized program visualization tools have been built and evaluated to observe and explore program execution in a visual form [11].

Computer-based technologies for presenting program construction examples are less explored. For many years, the state-of-the-art approach for presenting worked code examples in online tools was simply interleaving code with comments [1, 12, 13]. More recently, this approach has been enhanced with multimedia by adding audio narrations to explain the code [14] or by showing video fragments of code screencasts with the instructor's narration being heard while watching code in slides or an editor window [5, 6]. However, both ways support *passive* learning, which is the least efficient approach from the perspective of the ICAP framework [15][1]

An attempt to make learning from program construction examples *active* was made in the WebEx system, which allowed students to interactively explore instructor-provided line-by-line comments for program examples via a web-based interface [4]. More recently, several projects [6, 7, 8] augmented examples with simple problems and other constructive activities to elevate the example study process to the *interactive and constructive* levels of the ICAP framework, known as the most pedagogically efficient.

A good example of a modern interactive tool for studying code examples is the PCEX system [8]. PCEX (Program Construction EXamples) was created in the context of an NSF Infrastructure project (https://cssplice.org) with a focus

[1]The ICAP framework differentiates four modes of engagement: *interactive, constructive, active, and passive.*

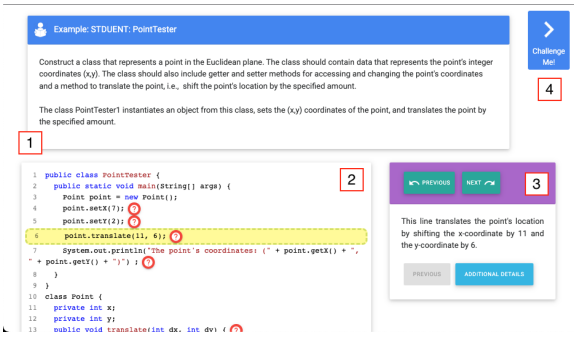

Figure 1: Studying a code example in the PCEX system: 1) title and program description, 2) program source code with lines annotated with explanations, 3) explanations for the highlighted line, 4) link to a "challenge" - a small problem related to the example.

on broad reuse and has been used by several universities in the US and Europe in the context of Java, Python, and SQL courses. PCEX interface (Figure 1) provides interactive access to traditionally organized worked examples, i.e., code lines augmented with instructor's explanations. Separating explanations (Figure 1-3) from the code (Figure 1-2), allows students to selectively study explanations for code lines they want. Explanations are provided on several levels of detail, so more details could be requested if the brief explanation is not sufficient (Figure 1-3).

Since line-by-line multi-level example explanations offered by PCEX is currently the most detailed approach for explaining worked examples, we selected the code example structure implemented by PCEX as the target model for our authoring tool introduced in the next section. The tool produces code augmented with line-by-line explanations on several levels of detail. The resulting example could be directly uploaded to PCEX or exported in a system-independent format to be uploaded to other example exploration systems such as WebEx [4].

2.2. Use of LLMs for Code Explanations

Multiple researchers have explored code summarization [16] and explanations using transformer models [17, 18], abstract syntax trees [19], and Tree-LSTM [20]. With the announcement of ChatGPT, several research teams explored the use of LLM for code explanations using GPT 3 [21, 22, 23], GPT 3.5 [22, 24, 25], GPT 4 [24], OpenAI Codex [26, 20, 22], and GitHub Copilot [25]. Table 1 presents a brief summary of the most important prior work.

In prior work, LLMs were used to generate explanations at different levels of abstraction (line-by-line, step-by-step, and high-level summary). Sarsa et al. [26] observed that ChatGPT can generate better explanations at low-level (lines). Li et al. [24] used the result of specific-to-general generated explanations as one of the inputs to their LLM solver, trying to solve competitive-level programming problems more efficiently. A novel research [21] tried to understand how non-experts approach LLMs. They have identified common mistakes and provided advice to tool designers.

The explanations and summaries generated by these LLMs were evaluated primarily by authors [26], students [22, 23], and tool users [25]. Sarsa et al. [26] reported a high correct ratio for generated explanations with minor mistakes that can be resolved by the instructor or teaching assistant.

Source	Goal	LLM(s)	Type of Explanations	Evaluation
[25]	Provide explanations for a code fragment selected in the IDE	GPT 3.5	Explain the selected code	Interview with students, teachers, and bootcamp tutors
[23]	Scaffold student's ability to understand and explain code	GPT 3	Explain the intended purpose of a function	Compare ChatGPT explanations with student/peer explanations
[24]	Given the problem description and expert solution, ChatGPT is prompted to generate explanations	GPT 3.5 vs GPT 4	Program summary, used algorithm, step-by-step solution description, time complexity, etc	Generated explanations were evaluated by the human programming expert who authored the "oracle" solution
[22]	Generate specific explanation, summary, and concepts for a given code snippet	GPT 3 and Codex	line-by-line explanations, list of important concepts, high-level summary of the code	Students' ratings of explanations, and their utility time/count
[26]	Help introductory programming course teachers by creating programming exercises + test cases, and code explanations	Codex	step-by-step explanation, problem-statement-like description, high-level description	Internal evaluation, measuring the percentage of code being explained

Table 1
Prior works in using LLMs (ChatGPT/Codex) to generate code explanations.

Students rated LLM-generated explanations as useful, easier, and more accurate than learner–sourced explanations [23].

Prompt, as an essential part of communication, directly influences the LLM's performance. A verbose prompt will limit the LLM's ability to utilize its knowledge [20]. Iterative prompts have been proven to perform well [21]. In terms of code explanation, providing the source code and expected outcome is essential. Adding input/output examples can help generate better explanations. Although LLMs such as ChatGPT can understand the natural language very well, the researchers suggested writing the prompt as writing a code: following a structure and marking different parts of the prompt [21]. If possible, it is better to control the randomness of LLMs responses (for instance, adjusting the temperature to a lower value, perhaps 0). Producing useful prompts requires some level of expertise - as Zamfirescu-Pereira and colleagues [21] observed, non-experts have misconceptions about LLMs and frequently struggle to come up with a well-formed prompt. Researchers believe that LLMs can be beneficial in environments where humans and AI can work together, where AI performs tasks known to be time-consuming for humans, while the human performs the expert evaluation and adjusts the responses generated by AI [27].

3. Prompt Tuning and Internal Evaluation

Following the majority recent work on generating code explanations, we choose ChatGPT as the target LLM to generate code explanations. ChatGPT provides an easy-to-use API and an affordable pricing model. Adding ChatGPT to an application is not a straightforward process and requires careful planning. The key part of this process is crafting a prompt, which requires multiple trials. Following the suggestions in the previous work [28, 25], the authors used an internal evaluation process to engineer a prompt that produces high-quality explanations.

To shorten the prompt design process, we adopted several design decisions that were shown to be effective in previous work: assigning a role to ChatGPT [27], avoiding verbosity [20], repetition [21], prompt that looks like code [21], and defining the expected output format [21]. However, some design decisions not evaluated previously were not evident, so we had to use an internal evaluation process to select the best performing option. The questions answered through the evaluation included the following: 1) Does the presence of a program description in the prompt result in better explanations? 2) Does iterative prompting perform better than a single prompt, and if so, how many iterations are sufficient to have a good explanation? 3) Does adding line inclusion/exclusion criteria in the prompt help ChatGPT to select or ignore lines in generating an explanation? To answer these questions, we formally compared ChatGPT-generated explanations through an independent rating performed by three authors of the paper.

Since we started from previously explored prompting techniques, the first version of our prompt was reasonably close to our final prompt. At the first stage of the process, we made a few small corrections of the prompt based on observations. First, we observed that ChatGPT cannot associate the line number with the line correctly. To address this issue, we marked each line with its line number. We also observed that sometimes, with iterative prompting, ChatGPT generates duplicate explanations. Hence, in our iterative prompts, we asked ChatGPT to generate explanations that are new. Figure 2 shows the final version of the prompt that we used with the *ChatGPT gpt-3.5-turbo/16k* model (*temperature=0*) using the OpenAI API for our internal and external evaluations.

Selecting Examples for Evaluation: We randomly selected eight Java examples with different difficulty levels (string operation, array, loop, and object-oriented programming) from the PCEX repository for the study. Selected examples include:

- *Initials*: Extracting initials from full name.
- *JAdjacentDuplicates*: Checks whether a sequence of numbers contains adjacent duplicates.
- *JArrayIncrementElements*: Increments all elements of the array by 1.
- *JArrayMax*: Finds the maximum value in an array.
- *JPrintDigitsReverse*: Prints the digits of an integer from right to left.
- *JSearchArrayValues*: Search for values from one array in another.
- *JSmallestDivisor*: Smallest divisor of a positive number.
- *PointTester*: Translate 2-dimensional coordinates.

Including/Excluding Program Description: We hypothesized that adding a program description for the prompt adds

Role (System):

You are a professor who teaches computer programming.

Role (User) - Iteration #1:

Given the following program description and accompanying source code, identify and explain lines of the code that contributes directly to the program objectives and goals. **[inclusion criteria]** **[exclusion criteria]**

When considering each identified line, ensure explanations provide the reasons that led to the line inclusion, prioritizing them based on their relative importance while also preventing any unnecessary duplication or repetition of information.

Program Description:
[program description]

Program Source Code:
The line number is defined as /*line_num*/ at the start of each line.

'''java
/*1*/[program lines]
'''

Output format:
Reply ONLY with a JSON array where each element, representing a "line of code," includes "line_num" and an "explanations" array. For example:
'''json
[{ "line_num": "2", "explanations": ["explanation ...", "explanation ...", ...] }, ...]
'''

Role (User) - Iteration #2:

Update your explanations with more insightful and complementary YET COMPLETELY new explanations. If you missed a line, this is the time to include them.

Role (User) - Iteration #nth:

Please repeat that once more.

Figure 2: ChatGPT Prompt Template. ChatGPT (is given the "professor" role) is prompted iteratively.

	with desc		without desc
Examples	R_2	R_3	R_2
Initials	88.8%	96.0%	90.7%
JAdjacentDuplicates	93.6%	99.0%	86.7%
JArrayIncrementElements	40.0%	–	93.3%
JArrayMax	85.7%	–	83.8%
JPrintDigitsReverse	57.1%	–	–
JSearchArrayValues	90.0%	71.4%	93.0%
JSmallestDivisor	46.3%	–	86.3%
PointTester	91.6%	–	73.3%

Table 2
Cosine similarity between rounds of explanations ($R_{n=2}$ = $cosine\_sim(R_n, R_{n-1})$) with and without including program description.

information for ChatGPT to produce better explanations, but we were also concerned that it could confuse ChatGPT. To compare the quality of the explanation with and without description, the evaluators checked the explanations for the following: 1) correctness, 2) relevance to the given program description (when present), 3) presence of new information in the 2^{nd} round compared to the 1^{st} round, 4) presence of hallucinations when the program description is not present, 5) whether the 2^{nd} round with program description in the prompt had more information than without description. Both Correctness and Relevance were binary ratings. For example, given an explanation *This line initializes a variable 'fullName' and assigns it the value 'John Smith'. The 'fullName' variable stores the full name of the person whose initials are to be printed.* for the line of code *String fullName = "John Smith";* was rated as "correct" and "relevant" by one rater.

	Round 1		Round 2		
	*C	**R	*C	**R	***A
Min	98.46%	44.62%	97.53%	37.04%	32.10%
Max	100.00%	70.77%	100.00%	68.75%	39.51%
Average*	99.23%	55.38%	98.77%	46.91%	35.80%

Table 3
Internal evaluators rating, when program description is present in the prompt, *Correctness, **Relevance to program description, ***Additional information compared to 1st round.

	Round 1		Round 2		
	*C	**H	*C	**H	***A
Min	93.98%	0.00%	92.94%	0.00%	41.18%
Max	100.00%	4.82%	100.00%	5.88%	55.29%
Average*	96.99%	2.41%	96.47%	2.94%	48.24%

Table 4
Internal evaluators rating, when program description is not present in the prompt, *Correctness, **Explanation contains hallucinations, ***Additional information compared to 1st round.

We, as internal evaluators, rated the correctness of the explanations higher in rounds 1 and 2 of the ChatGPT generation when the program description is present (R_1 = 99.23%, R_2 = 98.77%), as summarized in Table 3. As an interesting example, when observing the ratings that we used to compare the amount of new information generated in round 2 (compared to round 1), we observed that more information is generated without program description (48.24%) than with description (35.80%) as summarized in Table 4. The explanations generated when the program description is not present had additional information compared to when it is present as shown in Figure 3. This validates previous findings that comprehensive prompts limit LLM's ability to utilize their knowledge [20]. Furthermore, when the program description is present, the authors selected the 2^{nd} round of explanations for the external evaluation because students relate better with the program and it is also rated higher for correctness. In the conditions that we did not include the program description in the prompts, we were interested to know to what extent ChatGPT may hallucinate. We observed hallucinations 2.94% *on average* when considering prompts without program description, which could be attributed to greater information generation in round 2. Given this tendency to hallucinate when generating explanations with prompts do not include problem descriptions, we decided to use the explanations generated with prompts that include program descriptions.

Assessing Multi-Round Prompting: In this step, we assessed whether prompting ChatGPT iteratively to provide an explanation (see Figure 2) results in additional explanations. When the program description was present in the prompt, only 3 out of 9 examples had additional explanations compared to none when not included (Table 2). Explanations generated in the 3^{rd} round were either minor wording changes (high cosine similarity) or included explanations for unnecessary lines (closing bracket for main method and class). Qualitatively assessing explanations generated in

Program description: Write a program that finds the maximum value in an array.

Line #	Line Content	With Description 2nd Round	Without Description 2nd Round
3	int[] values = { 5, -4, 78, 95, 12 };	This line initializes an array called 'values' with the given values. The array 'values' represents the set of numbers from which the maximum value needs to be found.	This line initializes an array called 'values' with the given values. The array 'values' holds the numbers to be evaluated to find the maximum value. **The values in the array can be modified or expanded to include any set of numbers.**

Figure 3: Example of line explanations which the 2^{nd} round of explanation when the program description is not present had additional information compared to when present.

the 2^{nd} round, they included additional explanations or improved wordings. The number of additional explanations or improvements was not consistent among the examples in the 2^{nd} round, but on average, in 35.80% of lines (Figure 3) when the program description is present in the prompt, and in 48.24% of lines (Figure 4) when not, additional information was reported by the evaluators. Based on these findings, we decided to adopt a two-round prompting option for WEAT and used this option in the external evaluation process. We summarize our results in Table 2.

Assessing Inclusion/Exclusion Criteria: A program description can provide a rich context for identifying and explaining lines of code. However, ChatGPT may sometimes include an unnecessary line or exclude a necessary one from the explanation. Initially, we assumed that directly adding inclusion/exclusion criteria in the ChatGPT prompt can address this issue. However, evaluating this option internally, the authors observed that it resulted in less than 1% new lines inclusion and around $4-6\%$ of lines exclusion. When these criteria are present in the prompt, ChatGPT ends up having unnecessary rounds of explanations. Sometimes ChatGPT falls into a loop where it flips wordings between each round. Since the author can review and ignore the explanations for a specific line in the authoring interface, we decided not to use Inclusion/Exclusion criteria in the prompt.

4. Evaluation

We used two different approaches to evaluate the best-performing prompt, which we tuned through an internal evaluation process described above. In our previous work [29] we used several formal metrics to compare the explanations generated by ChatGPT with the explanations produced for the same code lines by domain experts and students in programming classes. In this paper, we report the results of a user-centic evaluation, in which two categories of target users compared the explanations generated by ChatGPT with the explanations created by domain experts for the same worked examples. Unlike some earlier studies that used beginner students to evaluate ChatGPT explanations, we used undergraduate students and senior graduate students (teaching assistants). The reason for adding teaching assistants as another category of users in the study is that in our authoring system, direct users of the ChatGPT explanation are not prospective *consumers* of explanations, but prospective *authors*. While the opinion of undergraduate students as consumers is important, in our human-AI collaborative authoring approach, authors have the option to edit the generated explanation before it will reach the consumers. It is important to understand how satisfactory the generated explanations appear to the prospective authors since their perception of quality impacts the amount of their work: poor explanations will require a lot of editing, while good explanations could be accepted as-is or with minimal changes.

To support these evaluation needs, we recruited 15 evaluators, of which 6 were graduate students doing research on computing education and 9 were undergraduate students who just completed an advanced Java programming class. Graduate students selected for the study usually serve as teaching assistants or instructors in programming classes where supplementary content development is their major responsibility. For brevity, we refer to them as *authors* in our analysis. While these authors are primary users of the generated explanations, the opinion of advanced undergraduate students was also important for us since they are frequently involved in learning content production through "learner-sourcing" [9, 30]. To distinguish them from the true authors, we refer to them as *students*. The participants had to provide their responses through an evaluation form. The evaluation was estimated to take one hour to complete. The participants received a $20 Amazon gift card as compensation.

The evaluation form included 8 examples introduced above. For each example, the form included a program description and the example code. For each line of each code example, it listed an explanation generated for this line by ChatGPT and by an expert. The participants had to rate both explanations for a given line of code and compare them. The order of ChatGPT and expert explanations for a given line of code was randomized, and the evaluators did not know which explanation was generated by ChatGPT or the expert. "Expert explanations" were extracted from real examples used in the PCEX system [31]. These explanations were originally authored by instructors and teaching assistants and refined through several years of classroom use.

To evaluate the explanations, the participants had to rate to what extent each explanation is *complete* and which is *better*. We defined a *better* explanation as "providing more information, going deeper, better connecting to programming concepts". However, we did not provide the definition of *complete explanation*. Leaving an open–ended coding task would help us understand the participants' use of their judgment on the completeness of the generated explanations.

More specifically, participants had to rate the two explanations with the following metrics (refer Figure 4):

1. *Explanation 1 is sufficiently complete*: Not complete (0), Complete (1), Very complete (2)
2. *Explanation 2 is sufficiently complete*: Not complete (0), Complete (1), Very complete (2)
3. *Which explanation is better?* Both are the same (0), Explanation 1 is better (1), Explanation 2 is better (2)

From the collected responses, we excluded lines that only ChatGPT or only expert explanations but not both. In these cases, the evaluators generally rated the explanations as better without comparison with a missing counterpart explanation. Altogether, there were 18 lines that were explained by ChatGPT but not by the expert, and 5 lines that were explained by the experts but not ChatGPT. Looking closer, we observed that 4 of the 5 missing lines of ChatGPT were in the *PointTester* example, which included class definition, object instantiation, and instance variable definition. We are not aware why the expert did not explain these lines, but we assume these lines are either mentioned in explanations generated for other lines or they do not provide important

Line #	Line Content	Explanation 1	Explanation 2	Explanation 1 is sufficiently complete	Explanation 2 is sufficiently complete	Which explanation is better?
Program Description: Construct a program that prints the initials of the name "John Smith".						
1	public class Initials					
2	{					
3	public static void main(String[] args)					
4	{					
5	String fullName = "John Smith";	This line declares and initializes a variable 'fullName' of type String with the value 'John Smith'. The 'fullName' variable stores the full name of the person whose initials are to be printed.	To store the name so that we can extract the initials	Very complete (2)	Complete (1)	Explanation 1 is better (1)

Figure 4: In this figure we present an instance of the evaluation form, with the filled-in responses from a participant. The participant indicates whether the explanation is complete and the explanation they find better. This was deployed in the form of worksheets for quick access and response collection.

information toward understanding the program. Although the program description had related wordings, there were missed by ChatGPT: "*Construct a class* that represents... The class should contain *data that represents the point's integer coordinates(x, y). ...* The class *PointTester instantiates an object from this class*, sets the (x, y) coordinates of the ...". Conversely, in 14 out of 18 of these lines, ChatGPT unnecessarily explained class, main method definition, and closing brackets (class, method, loop, and condition). The other 4 lines were informative and useful. This can support the importance of having inclusion criteria in the prompt.

For the remaining 45 lines of code, we observed from the evaluators' ratings for the question "Explanation 1 is sufficiently complete?" or "Explanation 2 is sufficiently complete?" that ChatGPT explanations were rated as 0.59% (not complete), 21.04% (complete) and 78.37% (very complete) compared to Expert explanations as 6.96% (not complete), 56.44% (complete), and 36.59% (very complete). In response to the question "Which explanation is better?", evaluators selected ChatGPT as the better explanation in 53.93% of lines, compared to experts (20.59%); and in the rest of the lines (25.48%) both were rated the same. Our calculations of the inter-rater reliability for the ratings of the question "Which explanation is better?" using Fleiss-Kappa gave us 0.182, $p < 0.01$ score of agreement. This can be interpreted as "slight agreement" based on the 2-raters/2-categories table. Given that Fleiss-Kappa is a chance-corrected coefficient, it can be interpreted as a better agreement due to the high number of subjects (45 lines of code by 15 evaluators) [32].

We observe that the students did not rate ChatGPT explanations incomplete at all with their 13.33% and 86.67% ratings being that ChatGPT explanations are complete and very complete, respectively. The authors also rated ChatGPT explanations as complete (32.59%) or very complete (65.93%). Hence, a majority of authors and students find ChatGPT explanations complete, as shown in Figure 5. In terms of comparing the explanations for which is better, 51.11% and 58.15% of students and authors, respectively, find that the explanations of ChatGPT are better for the given lines of code. A direct comparisons of two options, based on the question "which explanation is better (ChatGPT vs Expert)?", is presented in Figure 6. Given that the assessment was performed using blind rating, this is an encouraging result for the use of generative AI for authoring tools.

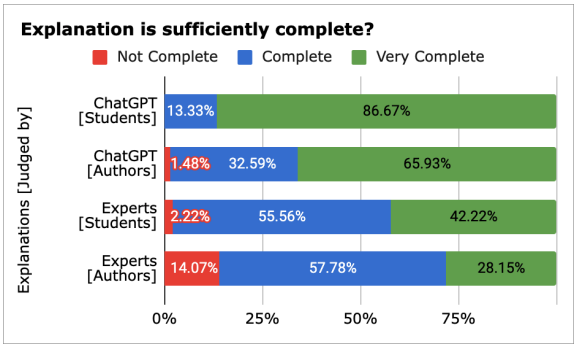

Figure 5: ChatGPT and Expert's explanations being judged by students and authors in terms of completeness.

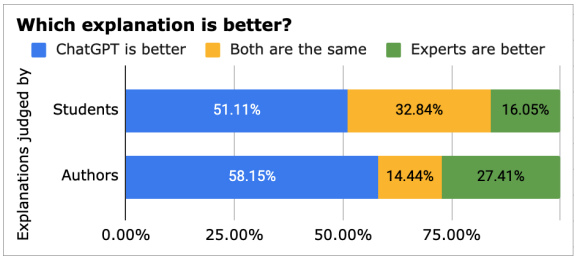

Figure 6: ChatGPT and Expert's explanations being judged by students and authors in terms of which one is better.

5. Discussion

The results of our analysis of participant evaluation responses show that both students and authors overwhelmingly rate ChatGPT explanations as "very complete" or at least "complete". The comparison also reveals that at average, both categories of users prefer ChatGPT explanations over expert explanations. However, in a sizeable fraction of cases, experts' explanations were judged to be better than ChaGPT

Taken together, these results provide good empirical evidence in support of our work on collaborative human-AI authoring of worked examples. On one hand, after prompt tuning, ChatGPT was able to produce good quality code explanations. On the other hand, in a good number of cases, the explanations produced by experts were better. In this context, a two-step process where ChatGPT start by generating explanations and humans edit the results, when necessary, could be "the best of both worlds". Moreover, given that in the majority cases ChatGPT explanations were equal to or better than expert explanations, such edits might

be necessary only in about one-third of the cases or less, making the collaborative authoring very efficient. Based on the results of this study, we are now working on a new version of Worked Examples Authoring Tool (WEAT) that includes options for editing and reusing low–level explanations generated by ChatGPT for instructors.

6. Limitations

As the first step towards this important goal, our work has several limitations. First, the scale of our evaluation was relatively small. Since we targeted prospective authors as users in our evaluation process, we were able to recruit only 15 qualified subjects. Furthermore, within the time allocated for the study, the subjects were able to process only eight worked examples. Although we attempted to broadly vary the topics and difficulty of selected examples to achieve sufficient generalizability of the results, a larger-scale study with a broader variety of examples might be necessary to obtain deeper insights. We plan to carry out such a study in our future work.

Although the use of the same best-performing prompt to generate explanations for examples of different difficulties was an important design decision to explore the generalizability of the approach, it might be possible that different prompts will perform best for examples of different difficulties. We will explore this opportunity in the next round of our work.

We also observed that for some lines of code in our dataset, experts, ChatGPT, or both choose to provide no explanations. In the current study, these lines were excluded from the evaluation as a meaningful comparison was not possible. However, choosing whether to explain a specific line or not is an important decision, and the current study did not assess who is making better decisions about skipping lines, ChatGPT or experts. This aspect requires further investigation. In our next study, we plan to ask participant evaluators to specify whether each line of code needs an explanation or not.

Another potential limitation of the study was the lack of a formal definition of what a "complete" explanation means during external evaluation. We let the participants decide how to rate completeness, since it is a personal decision which editors should make when deciding whether to update generated explanation or not. Although it was a natural thing to do, it could have decreased the agreement between the evaluators. In our future work, we will see whether the agreement could be increased by defending correctness and completeness ratings more formally.

Finally, an aspect of human-AI collaboration not explored in this study is the value of keeping our engineered prompt open to the authors to change. The existing research reviewed above demonstrates that users unfamiliar with LLM are unable to produce well-performing prompts [21]. However, most instructors and Teaching Assistants (TAs) in programming courses are computer scientists with graduate-level training. We expected that some fraction of these users could benefit from the ability to change the prompt and leave this option open. However, this assumption has to be explored. We hope that a study that engages real instructors or TAs in producing worked examples for their course might provide interesting data on end-user work with a prompt. The ultimate way to address these limitations and collect valuable information is to run a multi-semester-long study engaging instructors to use the tool to produce explanations. Such a study will also enable us to assess the quality of explanations produced through human-AI collaboration and their value to students in introductory programming classes.

7. Conclusion

In this paper, we report the results of our work in developing a worked example authoring tool that utilizes ChatGPT for the automatic generation of line-by-line code explanations. The idea of the tool is to allow humans and AI to collaborate in the process of authoring worked examples. To the best of our knowledge, this is the first attempt to produce worked examples through human-AI collaboration. Our work supports findings by other researchers and provides empirical evidence on the value of using ChatGPT to generate line-by-line code explanations. Through an external evaluation, this work also compared the generated explanations and human expert explanations.

8. Acknowledgments

We thank the organizers and mentors of the LearnLab Summer School 2023 for bringing us together and supporting our work on this project.

References

[1] M. C. Linn, M. J. Clancy, The case for case studies of programming problems, Commun. ACM 35 (1992) 121–132.
[2] H. M. Deitel, P. J. Deitel, C How to Program, 2nd Edition, Prentice Hall, New York, 1994.
[3] A. Kelley, I. Pohl, C by Dissection : The Essentials of C Programming, Addison-Wesley, New York, 1995.
[4] P. Brusilovsky, M. V. Yudelson, I.-H. Hsiao, Problem solving examples as first class objects in educational digital libraries: Three obstacles to overcome, Journal of Educational Multimedia and Hypermedia 18 (2009) 267–288.
[5] R. Sharrock, E. Hamonic, M. Hiron, S. Carlier, Codecast: An innovative technology to facilitate teaching and learning computer programming in a c language online course, Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (2017).
[6] K. Khandwala, P. J. Guo, Codemotion: expanding the design space of learner interactions with computer programming tutorial videos, Proceedings of the Fifth Annual ACM Conference on Learning at Scale (2018).
[7] J. Park, Y. H. Park, J. Kim, J. Cha, S. Kim, A. H. Oh, Elicast: embedding interactive exercises in instructional programming screencasts, Proceedings of the Fifth Annual ACM Conference on Learning at Scale (2018).
[8] R. Hosseini, K. Akhuseyinoglu, P. Brusilovsky, L. Malmi, K. Pollari-Malmi, C. Schunn, T. Sirkiä, Improving engagement in program construction examples for learning python programming, International Journal of Artificial Intelligence in Education 30 (2020) 299–336.
[9] I.-H. Hsiao, P. Brusilovsky, The role of community feedback in the student example authoring process: an

evaluation of annotex, British Journal of Educational Technology 42 (2011) 482–499.

[10] M. Hassany, P. Brusilovsky, J. Ke, K. Akhuseyinoglu, A. B. Lekshmi Narayanan, Human-ai co-creation of worked examples for programming classes, in: 5th Workshop on Human-AI Co-Creation with Generative Models (HA-GEN 2024) at IUI 2024, volume 3660, CEUR, 2024. URL: https://ceur-ws.org/Vol-3660/paper16.pdf.

[11] J. Sorva, V. Karavirta, L. Malmi, A review of generic program visualization systems for introductory programming education, ACM Trans. Comput. Educ. 13 (2013) 15:1–15:64.

[12] A. Davidovic, J. R. Warren, E. Trichina, Learning benefits of structural example-based adaptive tutoring systems, IEEE Trans. Educ. 46 (2003) 241–251.

[13] B. B. Morrison, L. E. Margulieux, B. Ericson, M. Guzdial, Subgoals help students solve parsons problems, Proceedings of the 47th ACM Technical Symposium on Computing Science Education (2016).

[14] B. Ericson, M. Guzdial, B. B. Morrison, Analysis of interactive features designed to enhance learning in an ebook, Proceedings of the eleventh annual International Conference on International Computing Education Research (2015).

[15] M. T. H. Chi, J. Adams, E. B. Bogusch, C. Bruchok, S. Kang, M. Lancaster, R. Levy, N. Li, K. L. McEldoon, G. S. Stump, R. Wylie, D. Xu, D. L. Yaghmourian, Translating the icap theory of cognitive engagement into practice, Cognitive Science 42 (2018) 1777–1832.

[16] J. Phillips, D. Bowes, M. El-Haj, T. Hall, Improved evaluation of automatic source code summarisation, Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM) (2022).

[17] Y. Choi, C. Na, H. Kim, J.-H. Lee, Readsum: Retrieval-augmented adaptive transformer for source code summarization, IEEE Access 11 (2023) 51155–51165.

[18] H. Peng, G. Li, Y. Zhao, Z. Jin, Rethinking positional encoding in tree transformer for code representation, in: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2022, pp. 3204–3214.

[19] Y. Shi, M. Chi, T. Barnes, T. W. Price, Code-dkt: A code-based knowledge tracing model for programming tasks, ArXiv abs/2206.03545 (2022).

[20] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, T. F. Bissyandé, Is chatgpt the ultimate programming assistant – how far is it?, 2023.

[21] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, Q. Yang, Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI '23, Association for Computing Machinery, New York, NY, USA, 2023.

[22] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, J. Leinonen, Experiences from using code explanations generated by large language models in a web software development e-book, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 931–937.

[23] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, A. Hellas, Comparing code explanations created by students and large language models, 2023.

[24] J. Li, S. Tworkowski, Y. Wu, R. Mooney, Explaining competitive-level programming solutions using llms, 2023.

[25] E. Chen, R. Huang, H.-S. Chen, Y.-H. Tseng, L.-Y. Li, Gptutor: A chatgpt-powered programming tool for code explanation, in: N. Wang, G. Rebolledo-Mendez, V. Dimitrova, N. Matsuda, O. C. Santos (Eds.), Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky, Springer Nature Switzerland, Cham, 2023, pp. 321–327.

[26] S. Sarsa, P. Denny, A. Hellas, J. 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, Association for Computing Machinery, New York, NY, USA, 2022, p. 27–43.

[27] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D. C. Schmidt, A prompt pattern catalog to enhance prompt engineering with chatgpt, 2023.

[28] D. Zhou, N. Scharli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, O. Bousquet, Q. Le, E. H. hsin Chi, Least-to-most prompting enables complex reasoning in large language models, ArXiv (2022).

[29] A. B. L. Narayanan, P. Oli, J. Chapagain, M. Hassany, R. Banjade, P. Brusilovsky, V. Rus, Explaining Code Examples in Introductory Programming Courses: LLM vs Humans, Technical Report, AI4ED Workshop @ AAAI 2024, 2024.

[30] J. J. Williams, J. Kim, A. Rafferty, S. Maldonado, K. Z. Gajos, W. S. Lasecki, N. Heffernan, Axis: Generating explanations at scale with learnersourcing and machine learning, in: Proceedings of the Third (2016) ACM Conference on Learning @ Scale, ACM, 2016, pp. 379–388.

[31] R. Hosseini, K. Akhuseyinoglu, A. Petersen, C. D. Schunn, P. Brusilovsky, Pcex: Interactive program construction examples for learning programming, in: Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli Calling '18, Association for Computing Machinery, New York, NY, USA, 2018.

[32] J. Sim, C. C. Wright, The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements, Physical Therapy 85 (2005) 257–268.