Explaining Code Examples in Introductory Programming Courses: LLM vs Humans

Arun Balajiee Lekshmi Narayanan, Priti Oli, Jeevan Chapagain, Mohammad Hassany, Rabin Banjade, Peter Brusilovsky and Vasile Rus

February 14, 2024

Proceedings of Machine Learning Research 1:1–11, 2010 AAAI AI4Edu 24

Explaining Code Examples in Introductory Programming Courses: LLM vs Humans

Arun-Balajiee Lekshmi-Narayanan ARL122@PITT.EDU
Priti Oli POLI@MEMPHIS.EDU
Jeevan Chapagain JCHPGAIN@MEMPHIS.EDU
Mohammad Hassany MOH70@PITT.EDU
Rabin Banjade RBNJADE1@MEMPHIS.EDU
Peter Brusilovsky PETERB@PITT.EDU
Vasile Rus VRUS@MEMPHIS.EDU
University of Pittsburgh, PA, USA. 15260
University of Memohis, TN, USA, 38152

Abstract

Worked examples, which present an explained code for solving typical programming problems 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 explanations for many examples typically used in a programming class. In this paper, we assess the feasibility of using LLMs to generate code explanations for passive and active example exploration systems. To achieve this goal, we compare the code explanations generated by chatGPT with the explanations generated by both experts and students.
Keywords: Programming, Worked Examples, Code Explanations, ChatGPT

1. Introduction

Program code examples (known also as worked examples) play a crucial role in learning how to program (Linn and Clancy, 1992). Instructors use examples extensively to demonstrate the semantics of the programming language being taught and to highlight the fundamental coding patterns. Programming textbooks allocate considerable space to present and explain code examples. To make the process of studying code examples more interactive, CS education researchers developed a range of tools to engage students in the study of code examples. These tools include codecasts (Sharrock et al., 2017), interactive example explorers (Hosseini et al., 2020), and tutoring systems (Oli et al., 2023).

An important component in all types of program examples is code explanations associated with code lines or chunks. The explanations connect examples with general programming knowledge explaining the role and function of code fragments or their behavior. In textbooks, these explanations are usually presented as comments in the code or as explanations on the margins. The example explorer tools allow students to examine these explanations interactively (Hosseini et al., 2020). Tutoring systems, which engage students in explaining the code, use instructor explanations to assess student responses (Chapagain et al., 2022) and provide scaffolding (Oli et al., 2023). The explanations must be *authored* by instructors or domain experts, i.e., written and integrated into a specific system. As the experience of the last 10 years demonstrated, these explanations are hard to obtain. Being enthusiastic about sharing *the code* of their worked examples with others, instructors generally do not have time or patience to properly author *explanations* of their examples. Indeed, creating just one explained example could take 30 minutes even in the presence of authoring tools (Hosseini et al., 2020; Sharrock et al., 2017). As a result, the volume of worked examples available to students in a typical introductory programming class is low.

To address this *authoring bottleneck*, researchers explored *learner-sourcing*, that is, engaging students in the creation and review of explanations of instructor-provided code (Hsiao and Brusilovsky, 2011) and the automatic extraction of explanations from lecture recordings (Khandwala and Guo, 2018). In this paper, we explore the feasibility of human-AI collaboration in creating explained code examples. With this approach, the instructor provides the code for their favorite examples. The AI engine based on large language models (LLM) examines the example code and generates explanations for each code line. The explanations are reviewed and, if necessary, edited by the instructor.

To assess the feasibility of this approach, it is important to compare the code explanations produced by LLMs such as ChatGPT with explanations produced by humans. To use ChatGPT explanations in example explorer systems, we need to check how similar they are in language, semantics, and style used to code explanations produced by instructors and domain experts. To use ChatGPT explanations to assess student responses in tutoring systems, we need to assess how close these explanations are to the explanations produced by students when explaining the code. In this paper, we employ a range of analytical approaches to compare ChatGPT code explanations with explanations produced by both experts and students. Following a review of past work on using LLM for code explanations, we present the method and datasets used in the study and review the results.

2. Related Work: Use of LLMs for Code Explanations

Multiple researchers have explored code summarization (Phillips et al., 2022) and explanations using transformer models (Choi et al., 2023; Peng et al., 2022), abstract syntax trees (Shi et al., 2022), and Tree-LSTM (Tian et al., 2023). With the announcement of ChatGPT, several research teams explored the use of LLM for code explanations using ChatGPT 3 (Zamfirescu-Pereira et al., 2023; MacNeil et al., 2023; Leinonen et al., 2023), GPT 3.5 (MacNeil et al., 2023; Li et al., 2023; Chen et al., 2023), GPT 4 (Li et al., 2023), OpenAI Codex (Sarsa et al., 2022; Tian et al., 2023; MacNeil et al., 2023), and GitHub Copilot (Chen et al., 2023). These LLMs were used to generate explanations at different levels of abstraction (line-by-line, step-by-step, and high-level summary). Sarsa et al. (2022) observed that ChatGPT can generate better explanations at line-by-line level. Li et al. (2023) 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.

The explanations and summaries generated by these LLMs were mostly evaluated by authors (Sarsa et al., 2022), students (MacNeil et al., 2023; Leinonen et al., 2023), and tool users (Chen et al., 2023). Most recently, attempts have been made to compare code explanations generated by humans and LLMs. Sarsa et al. (2022) reported that students rated LLM-generated explanations as useful, easier, and more accurate than learner-sourced explanations (Leinonen et al., 2023). In this work, we attempted a more formal approach to compare the line-by-line code explanations generated by ChatGPT, students, and experts using a range of quantitative metrics. Our goal was to generate insights into the use of ChatGPT code explanations in the learning process.

3. Dataset Collection

To produce the evaluation dataset, we collected line-by-line explanations from three types of sources - experts, ChatGPT, and students - for four Java worked examples selected from the PCEX example exploration system (Hosseini et al., 2020). These examples were used in multiple Java classes and include line-by-line explanations produced by the instructors. The summary of the types of explanation collected is shown in Table 1 and the collection process is explained below. Appendix A shows sample explanations from different sources.

Expert Explanations: We used one set of expert explanations available at PCEX and collected the second set of line-by-line explanations from different experts.

Student Explanations: We performed a user study in which students of a Java programming course were asked to write explanations for each line of the code examples selected for the study. In total, we collected line-by-line explanations from 60 students.

ChatGPT Explanations: We performed a sequence of internal studies to build ChatGPT prompts, which can produce the most useful line-by-line example explanations. For the final evaluation, we selected three prompts that produced good explanations on three different levels of details. We used gpt-3.5-turbo to generate four sets of line-by-line explanations for each selected example using each of these prompts. To increase the diversity of explanations, the first set was generated with temperature value 0 and three additional sets were generated with temperature value 1 each time clearing the history.

Simple Prompt: The *Simple prompt* used in the study included the code of the worked example and the following instruction: *"Provide a line-by-line self-explanation for each line of code in the Java program above"*. The explanations generated by this prompt are concise and elicit the goal of each line of code quite well.

Advanced Prompt: The *Advanced prompt* used both the problem statement and the example code along with more elaborate instructions for ChatGPT. The role of "a professor who teaches computer programming" is assigned to the system to provide a context. The prompt also asked ChatGPT to provide reasons why the line needs to be explained. We observed that ChatGPT cannot always associate line numbers correctly, so each line in the program source code was annotated with its associated line number. An output format was defined to process the results digestible by our automation script. Prompt details are provided in Figure 1 in Appendix B (Iteration #1).

Extended Prompt: To obtain the most elaborate ChatGPT explanations, we used *Extended prompt*, which requested ChatGPT to further enhance the explanations generated by the Advanced prompt (Iteration #2 in Figure 1). To finalize the version of the prompt used in the study, we tried several approaches to improve the already generated explanations and observed that asking ChatGPT to extend the explanation once produces the best results. Asking for a second extension does not provide a distinguishable improvement.

Explanation Type	N	Definition
Experts	2	Source Code Line-by-Line Explanations by Experts
Students	60	Source Code Line-by-Line Explanations by Students
Simple Prompt (S)	4	ChatGPT Explanations with simple prompt (Section 3)
Advanced Prompt (A)	4	ChatGPT Explanations with advanced prompt (Figure 1)
Extended Prompt (E)	4	ChatGPT Explanations with extended prompt (Figure 1)
ChatGPT	12	Aggregated representation for all ChatGPT prompts

Table 1: A summary of explanation sources used in the study.

4. Evaluation Metrics

Lexical Metrics: We report the *lexical diversity* and *lexical density* of the generated explanations to assess the richness, informativeness, and conciseness of the generated text (Johansson, 2008). *Lexical diversity* is the range of variety of distinct words or vocabulary used within a specific text. *Lexical density* refers to the measure of the variety of different lexical words present in a text, including nouns, adjectives, verbs, and adverbs, which collectively contribute to the overall meaning of the text. A recent paper considers the use of lexical diversity as a metric (Cegin et al., 2023) to compare content generated by humans and ChatGPT. We also report on the total number of tokens on each explanation to provide more insight on the comparison of the lexical features across each source.

Readability Metrics: We consider 3 popular metrics (Denny et al., 2020, 2021), namely, Flesch-Kincaid Grade Level, Gunning Fog and Flesch Reading Ease. These metrics estimate the grade level expectation of the text such as the years of formal education required to read the text and the ease of reading a piece of text respectively. We use the TextDescriptives (Hansen et al., 2023) package in Python [1] to calculate these scores.

Similarity Metrics: We use four evaluation metrics: Character-based metric chrF (Popović, 2015), Word-based metric METEOR (Banerjee and Lavie, 2005), and Embedding-based metrics BERTScore (Zhang et al., 2019) and Universal Sentence Encoder (USE) (Cer et al., 2018) to compare explanation generated by different sources. chrF (character n-gram F-score) measures the character-level matching between the reference text and the machine-generated text considering both precision and recall. METEOR considers the similarity between words and assesses word overlap between the two texts. BERTScore is an automated evaluation metric for text generation that assesses the similarity between candidate and reference sentences by comparing the contextual embeddings of individual tokens using cosine similarity. USE is a transformer-based model that transforms text into high-dimensional vectors, enabling the computation of similarity between two texts based on their vector representations. Haque et al. (2022) and Roy et al. (2021) have pointed out that METEOR, chrF (Popović, 2015), and USE (Cer et al., 2018) metrics are better aligned with human preferences of code summarization as these metrics assign partial credits to words. We also use BertScore to evaluate the generated explanation primarily due to its extensive use as a reliable measure for evaluating the faithfulness of LLMs (Ji et al., 2023). Consequently, traditional metrics like BLEU (Papineni et al., 2002) which solely rely on word overlap, are now considered outdated and are not included in our reporting.

1. https://hlasse.github.io/TextDescriptives/readability.html

5. Results

To compare the explanations produced by different sources, code explanations for all 33 explainable lines of four examples generated by each expert, each student, and each round of ChatGPT generation were merged in a single source document. The metrics were calculated for each of these sources. Table 2 reports the medians for each source *type* (i.e., all students, all experts, each ChatGPT prompt, and all ChaptGPT prompts together.

Lexical Metrics: As Table 2 shows, explanations produced by experts and ChatGPT are more than twice as longer than explanations produced by students (as measured by the number of tokens). Students also use considerably fewer unique words in their explanations (lexical diversity) hinting that their vocabulary is more narrow than the vocabulary of experts and ChatGPT. The length and lexical diversity of explanations generated by ChatGPT and experts varied, with *Simple prompt* generating the shortest, *Extended prompt* the longest explanations, and expert explanations positioned between *Advanced* and *Extended* prompts. An ANOVA analysis of the lexical diversity of the explanation generated by experts, ChatGPT, and students indicated statistically significant variations among the groups (F-statistic = 25.07, $p < 0.05$). The data also shows that explanations produced by students are not only shorter than those by experts and ChatGPT, but also have higher lexical density, suggesting that the students explain the code in a more "concentrated" way. However, the ANOVA analysis of lexical density indicates no significant difference (F-statistic = 2.5, p =0.08) between the explanations in terms of lexical density.

Source	N	Vocabulary	Lexical Density	# of Tokens	GF	FRE	FK
Experts	2	209.0	0.48	690.0	8.46	78.45	6.18
S	4	165.0	0.45	517.5	8.67	82.34	6.35
A	4	185.5	0.48	625.0	9.91	72.63	7.15
E	4	238.0	0.49	769.5	11.09	69.64	7.83
ChatGPT	12	179.5	0.48	625.0	8.99	75.41	6.69
Students	60	116.5	0.54	249.5	8.02	80.48	5.62

Table 2: Median lexical and readability metrics for different sources of explanations (FRE = Flesch-Reading Ease, FK = Flesch-Kincaid, GF = Gunning Fog)

Readability Metrics: One-way ANOVA revealed that the Gunning-Fog readability scores are significantly different across the sources of explanations (experts, students and ChatGPT) ($p < 0.001$). From posthoc comparisons, most significant differences were observed between explanations produced by *Extended prompt* and experts ($p = 0.0102$) as well as students ($p < 0.0001$). No significance was observed between experts and students ($p = 0.0848$). Overall, for all metrics, the ChatGPT explanations are relatively less readable (more technical) than those of experts, which are less readable than those of students (Table 2).

Similarity Metrics: We applied similarity metrics to calculate the similarity between the explanations provided by ChatGPT, students, and experts for each line of code. To calculate the similarity between two specific sources (e.g each student's explanation with each expert's explanation), we averaged the similarity values for all 33 lines of explanations between the two sources. To determine the overall similarity between students and experts, we computed the average similarity score for all pairs of student and expert explanations. Similarly, we calculated the similarity between ChatGPT and expert explanations, between ChatGPT and student explanations, and between different ChatGPT prompting strategies.

Table 3 shows that the explanations generated by ChatGPT exhibit a consistently higher average similarity score to the expert explanations as compared to those generated by students. This suggests that the explanations generated by ChatGPT are more closely aligned with expert explanations than with student explanations across all metrics. The results of a Mann-Whitney U-tests indicate a statistically significant level (F-statistic = 48.0, $p < 0.05$ for METEOR, F-statistic = 205.0, $p < 0.05$ for USE and F-statistics=288.0, $p < 0.05$ for BERTScore) of alignment between ChatGPT explanations and expert explanations when compared to the alignment between student explanations and expert explanations across these metrics. We observed that the explanation generated by *Simple* prompting strategy(S) aligned more closely with expert-level explanations as compared to *Advanced* and *Extended prompting* (A and E). However, all the prompting strategies demonstrated similar levels of similarity when compared to the explanations provided by the students. Furthermore, we observed that the explanations generated by ChatGPT with all the prompting strategies exhibit semantic alignment.

Reference	Source	chrF	METEOR	USE	BERTScore
Expert	ChatGPT(S)	0.32	0.28	0.54	0.89
Expert	ChatGPT(A)	0.31	0.27	0.49	0.709
Expert	ChatGPT(E)	0.32	0.28	0.48	0.712
Expert	ChatGPT (All)	0.32	0.27	0.50	0.75
Expert	Student	0.33	0.144	0.33	0.63
ChatGPT(S)	Student	0.22	0.21	0.34	0.60
ChatGPT(A)	Student	0.18	0.150	0.254	0.450
ChatGPT(E)	Student	0.18	0.151	0.255	0.458
ChatGPT(All)	Student	0.19	0.17	0.28	0.50
ChatGPT(S)	ChatGPT(A)	0.33	0.30	0.50	0.72
ChatGPT(S)	ChatGPT(E)	0.32	0.28	0.50	0.73
ChatGPT(A)	ChatGPT(E)	0.44	0.43	0.56	0.69

Table 3: Assessing alignment (larger is better) between sources of explanations

6. Conclusions and Future Work

In this work, our goal was to assess the feasibility of using ChatGPT to generate line-by-line code explanations to be used in worked-out examples in place of currently used expert explanations. To achieve this goal, we compared the ChatGPT explanations generated by different prompts with the student and expert explanations using a range of metrics. Our results indicate that the explanations generated by GhatGPT are lexically and semantically similar to the explanations generated by experts and could potentially resolve the authoring bottleneck. However, their lower readability level might be an obstacle for less-prepared students. We also observed a considerable difference between the explanations produced by students and explanations produced by experts and ChatGPT, which might affect the efficiency of both sources of explanation in active example tutors where student explanations are assessed by comparing them with expert explanations. Both issues require a deeper investigation, which we plan to perform through a user study.

References

Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

Ján Cegin, Jakub Simko, and Peter Brusilovsky. Chatgpt to replace crowdsourcing of paraphrases for intent classification: Higher diversity and comparable model robustness. *ArXiv*, abs/2305.12947, 2023. URL https://api.semanticscholar.org/CorpusID:258832868.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

Jeevan Chapagain, Lasang Tamang, Rabin Banjade, Priti Oli, and Vasile Rus. Automated assessment of student self-explanation during source code comprehension. In *The International FLAIRS Conference Proceedings*, volume 35, 2022.

Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. GPTutor: A ChatGPT-powered programming tool for code explanation. In *Artificial Intelligence in Education. Vol. 2*, pages 321–327, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-36336-8.

Yunseok Choi, Cheolwon Na, Hyojun Kim, and Jee-Hyong Lee. Readsum: Retrieval-augmented adaptive transformer for source code summarization. *IEEE Access*, 11:51155–51165, 2023. doi: 10.1109/ACCESS.2023.3271992.

Paul Denny, James Prather, and Brett A. Becker. Error message readability and novice debugging performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 480–486, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3387384. URL https://doi.org/10.1145/3341525.3387384.

Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. On designing programming error messages for novices: Readability and its constituent factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445696. URL https://doi.org/10.1145/3411764.3445696.

Lasse Hansen, Ludvig Renbo Olsen, and Kenneth C. Enevoldsen. Textdescriptives: A python package for calculating a large variety of metrics from text. *J. Open Source Softw.*, 8:5153, 2023. URL https://api.semanticscholar.org/CorpusID:257019557.

Sakib Haque, Zachary Eberhart, Aakash Bansal, and Collin McMillan. Semantic similarity metrics for evaluating source code summarization. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 36–47, 2022.

Roya Hosseini, Kamil Akhuseyinoglu, Peter Brusilovsky, Lauri Malmi, Kerttu Pollari-Malmi, Christian Schunn, and Teemu Sirkiä. Improving engagement in program construction examples for learning python programming. *International Journal of Artificial Intelligence in Education*, 30(2):299–336, Jun 2020. ISSN 1560-4306. doi: 10.1007/s40593-020-00197-0. URL https://doi.org/10.1007/s40593-020-00197-0.

I-Han Hsiao and Peter Brusilovsky. The role of community feedback in the student example authoring process: an evaluation of annotex. *British Journal of Educational Technology*, 42(3):482–499, 2011. doi: http://dx.doi.org/10.1111/j.1467-8535.2009.01030.x.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

Victoria Johansson. Lexical diversity and lexical density in speech and writing: A developmental perspective. *Working papers/Lund University, Department of Linguistics and Phonetics*, 53:61–79, 2008.

Kandarp Khandwala and Philip 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. URL https://api.semanticscholar.org/CorpusID:49304895.

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*. 2023. arXiv:2304.03938.

Jierui Li, Szymon Tworkowski, Yingying Wu, and Raymond Mooney. Explaining competitive-level programming solutions using LLMs, 2023. arXiv:2307.05337.

Marcia C. Linn and Michael J. Clancy. The case for case studies of programming problems. *Commun. ACM*, 35:121–132, 1992. URL https://api.semanticscholar.org/CorpusID:2856243.

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 *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, page 931–937, 2023. ISBN 9781450394314. doi: 10.1145/3545945.3569785. URL https://doi.org/10.1145/3545945.3569785.

Priti Oli, Rabin Banjade, Arun Balajiee Lekshmi Narayanan, Jeevan Chapagain, Lasang Jimba Tamang, Peter Brusilovsky, and Vasile Rus. Improving code comprehension through scaffolded self-explanations. In *Proceedings of 24th International Conference on Artificial Intelligence in Education, Part 2*, pages 478–483. Springer, 2023. URL ttps://doi.org/10.1007/978-3-031-36272-9_75.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

Han Peng, Ge Li, Yunfei Zhao, and Zhi Jin. Rethinking positional encoding in tree transformer for code representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3204–3214, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.210. URL https://aclanthology.org/2022.emnlp-main.210.

Jesse Phillips, David Bowes, Mahmoud El-Haj, and Tracy Hall. Improved evaluation of automatic source code summarisation. *Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, 2022. URL https://api.semanticscholar.org/CorpusID:256461175.

Maja Popović. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pages 392–395, 2015.

Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1105–1116, 2021.

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*, page 27–43, 2022. ISBN 9781450391948. doi: 10.1145/3501385.3543957. URL https://doi.org/10.1145/3501385.3543957.

Rémi Sharrock, Ella Hamonic, Mathias Hiron, and Sébastien Carlier. Codecast: An innovative technology to facilitate teaching and learning computer programming in a c language online course. *Proceedings of the Fourth ACM Conference on Learning @ Scale*, 2017. URL https://api.semanticscholar.org/CorpusID:9327165.

Yang Shi, Min Chi, Tiffany Barnes, and Thomas William Price. Code-DKT: A code-based knowledge tracing model for programming tasks, 2022. ArXiv:2206.03545.

Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. Is ChatGPT the ultimate programming assistant – how far is it?, 2023. arXiv:2304.11938.

J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian 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, 2023. ISBN 9781450394215. doi: 10.1145/3544548.3581388. URL https://doi.org/10.1145/3544548.3581388.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

Appendix A. Sample Data

Program: PointTester

line-num: 12

line-code: private int y;

Expert1: Every object of the Point class will have its own y-coordinate. Therefore, we need to declare an instance variable for the class to store the y-coordinate of the point. We declare it as int because we want to have integer coordinates for the point. Note that an instance variable is a variable defined in a class, for which each instantiated object of the class has a separate copy, or instance.

Expert2: The instance variables are declared as private to prevent direct access to them from outside the class. In this way, no unexpected modifications to a Point object's data are possible.

Expert3: We need to define a private integer variable y which belongs to class point. It represents the y-coordinate of the point.

S: This line declares a private integer variable named "y" to store the y coordinate of a point.

A1: This line defines a private instance variable 'y' of type int in the Point class.It contributes directly to the program's objective of storing the y-coordinate of the point.

E: This line declares a private instance variable 'y' of type int in the Point class to store the y-coordinate of a point.Declaring the y-coordinate variable is essential for keeping track of the point's position and contributes directly to the program's objective of storing the point's coordinates.

Student1: initialize a private value inside the point class with no value yet

Student2: Declares the private int variable y.

Student3: Creates a private int that can only be accessed by class Point called int y

.

.

.

Student59: private variable used to store the value entered into the value of the y coordinate.

Appendix B. Prompt Template

Below we present the template for one of our prompts used to generate ChatGPT line-by-line explanations to the source code of worked examples.

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.

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.

Figure 1: This ChatGPT Prompt template considers the case that ChatGPT could generate a better explanation with an additional "nudge" as observed above. In most cases, the generated explanations using the prompt at the second iteration produces richer explanation than the first iteration.