Enhancing Programming eTextbooks with ChstGPT Generated Counterfactual-Thinking-Inspired Questions[*]

Arun Balajiee Lekshmi-Narayanan[1,*,†], Rully Agus Hendrawan[1,*,†] and Venktesh V[2]

[1]*University of Pittsburgh, Pittsburgh, PA, USA*

[2]*Indraprastha Institute of Information Technology, Delhi,India*

Abstract

Digital textbooks have become an integral part of everyday learning tasks. In this work, we consider the use of digital textbooks for programming classes. Generally, students struggle with utilizing textbooks on programming to the maximum, with a possible reason being that the example programs provided as illustration of concepts in these textbooks don't offer sufficient interactivity for students, and thereby not sufficiently motivating to explore or understand these programming examples better. In our work, we explore the idea of enhancing the navigability of intelligent textbooks with the use of "counterfactual" questions, to make students think critically about these programs and enhance possible program comprehension. Inspired from previous works on nudging students on counter factual thinking, we present the possibility to enhance digital textbooks with questions generated using GPT–3.5.

Keywords

OpenAI ChatGPT, GPT–3.5, Large Language Models, Intelligent Textbooks, Program Comprehension, Critical Thinking, Question Generation,

1. Introduction & Related Work

Interactive textbooks have been explored extensively among the computer science education community, for many years [1], with the best examples of these being ELM-ART [2] for LISP programming; OpenDSA [3] an online reading platform for topics in Data Structures and Algorithms; Runestone [4] a platform to host interactive programming practice material, among other, suggesting that some of the most common eTextbooks for computer science education are on programming. In this work, we consider the enhancement of digital textbooks with the use of newer tools such as Large Language Models, as explored in the recent efforts on the use of Large Language Models to generate questions for intelligent textbooks [5].

Woodstock'22: Symposium on the irreproducible science, June 07–11, 2022, Woodstock, NY

[*]You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

[*]Corresponding author.

[†]These authors contributed equally.

arl122@pitt.edu (A. B. Lekshmi-Narayanan); ruhendrawan@pitt.edu (R. A. Hendrawan); venktesh.v@iiitd.ac.in (V. V)

https://a2un.github.io (A. B. Lekshmi-Narayanan); https://ruhendrawan.com/ (R. A. Hendrawan)

0000-0002-7735-5008 (A. B. Lekshmi-Narayanan); 0000-0001-8541-0305 (R. A. Hendrawan)

From prior work, we have looked at cases of support for program comprehension using question generation. Specifically, these approaches are directed at some of the following ideas, namely,

1. **Critical Thinking and Perspective Taking**: Promote a growth mindset [6] and foster robust critical thinking that nudges students towards a higher understanding. This might help students take a step back [7], and think slower and more deeply, encouraging perspective-taking.
2. **Syntax Analysis**: Encourage students to think based on specific errors and functional keywords within programs [8] about the syntax of the programs, deepening their understanding of syntactic elements.
3. **Goal-Oriented Analysis**: Focus on the steps and possibilities when tracing a program [9, 10], which is essentially about understanding the objective or the goal of the code and how it is achieved. Their work encourages students to think about what changes in the program would mean for its final output or goal.
4. **Problem-Solution Mapping**: Generate questions based on misconceptions [11]. The approach of asking why or why not a certain line in a program works encourages students to think about the problem that the specific line of code is solving. This helps them reverse engineer the code, i.e., map the solution (code) to the problem.
5. **Intrinsic Program Analysis**: Encourage students to deeply analyze the program by generating questions that are intrinsically tied to the program [12], while stimulating students to think about the program and its inherent aspects [13].
6. **Large Language Model Prompt-based QG**: While the focus of the question generation using LLM for intelligent digital textbooks in prior work [5] is similar to the current work, our work attempts to address the notion of prompting the LLM to an extent where the questions generated could be hierarchical, also specifically, counterfactual questions that allow room for alternative and creative mental constructs that support program comprehension.

In most cases, the question generation is not the key idea to enhance program comprehension. And in the cases where the program comprehension is supported by question generation [12, 13], the approaches have specific types of questions that they generate using fixed templates for the generation tasks using "in-filling" approach.

Additionally, Lehtinen and colleagues [13] construct the question types from prior framework, but these ideas do not necessarily consider an hierarchical approach towards a step-wise program comprehension. If we refer to the past work on program comprehension [14], we can proposition that questions that invoke students to think critically can also be hierarchical. Some other ideas that support this are the ICAP Framework [15]

Hence, our contributions in this work are three-fold:

1. *Systematically explore the use of prompt mechanism in the latest freely available ChatGPT (GPT–3.5) system to generate questions for a JAVA Program*
2. *Understand the relations between the ChatGPT (GPT–3.5) suggested categorization of these questions and those annotated by humans. Such an understanding could promote transparency in the process of LLM response to prompts.*

3. *Present a dataset of questions that are at varying levels in the hierarchy used to describe the program comprehension task. These questions could act as an enhancement to an existing intelligent, digital textbook on programming*

2. Counterfactual-Thinking Inspired Questions Generation

Building on existing coding challenges, we generate a set of questions inspired by previous works related to counterfactual thinking. These questions are then manually categorized into distinct themes, enabling a deeper understanding of prevalent trends and insights into the application of counterfactual thinking in problem-solving and creative tasks within programming.

2.1. Code Challenges

Practice exercises help novice programmers learn how to code. The process of learning to code involves understanding various programming concepts and practicing them until they become second nature. This is where practice exercises play a crucial role. These exercises provide an opportunity for novice programmers to apply the theory they have learned, strengthen their problem-solving skills, and learn from trial and error. Sample code challenges are categorized according to their functionality as follows:

1. Object and Arithmetic: Student Profile, Circle Area Calculator, Coordinate Shift, Shape Measurements
2. Repeated Calculation: Average Calculator, Bingo Board, Grade Calculator, Multiplication Table, Prime Checker
3. Comparisons and Rules: Place Name Comparator, Age Comparison, Phone Buyer, Bank Account

2.2. Question Generation with LLM

The efficacy of Language Learning Models (LLMs) is noteworthy [16]. If prompted correctly, an LLM can assist teachers in generating programming questions. We develop 5 prompts to generate questions for each code challenge. Every Prompt starts with the phrase *"In tabular file format with the following columns (LineNumber, LineCode, Question) generate counterfactual questions to make students critically think about the program from .. "*, except for a change with a few keywords as specified in Table 1.

2.3. Labeling Generated Questions

The classes used to label questions asked about a piece of code or a programming concept are:

1. **Syntax (S)**: The ability of code to compile successfully depends on whether it adheres to the syntax rules of the programming language. The concept of syntax in a programming language includes the proper use of brackets, punctuation like semicolons or colons, variable declaration, and so on. E.g., *"What would happen if we forgot to include the semicolon at the end?"*

Category	Prompt Keywords
Critical Thinking and Perspective Taking	*different perspectives or aspects*
Syntax Analysis	*syntactic perspectives or aspects*
Goal-Oriented Analysis	*such that final goal of the program is changed*
Problem-Solution Mapping	*could ask that have the same program as the solution*
Intrinsic Program Analysis	*based on the program and the answer to those questions lie within the program*

Table 1
List of various Prompts Keywords used to Generate questions of a certain category

2. **Programming Logic (PL)**: Logical comprehension and overall understanding of a piece of code or a program. The emphasis is not on specific syntax or language constructs, but rather on understanding how the pieces of the code fit together to create a functioning program. This could involve understanding the purpose of specific variables or functions, how control flow structures like loops or conditionals are used, or the logic behind a specific algorithm or data structure used in the code. In some cases, these questions could involve modifications or adaptations of existing code to meet new requirements or goals. E.g. *"What if we needed to read input from a file instead of from the user? How could we modify the code to achieve this?"*
3. **Goal-oriented (G)**: Achieving a desired result regardless of the specifics of the code. This category is more focused on the problem-solving aspect of programming, where the specific language or implementation details are secondary to the overall objective. E.g. *"What if we wanted to read the radius value from a file instead of user input?"*
4. **Miscellaneous (M)**: A catch-all category for questions that don't neatly fit into the other categories. This could involve questions about programming best practices, questions about specific development tools or environments, version control, debugging strategies, or questions about the broader principles and philosophies of software development. E.g., *"Why is the main method necessary in a Java program?"*

2.4. Thematic Categorization of Generated Questions

Once the set of questions is generated, the next step is to categorize them. The categorization phase is an iterative process that seeks to classify the generated questions based on distinct themes. This grouping process is designed to be flexible and iterative, allowing for the continuous refinement of categories as manifested themes or nuances are emerging during the analysis process. We focus on themes that are specific to programming tasks rather than broader aspects of problem-solving or creativity.

Each question is analyzed focused on its intent, the concepts it explores, and the specific aspects of programming it addresses. The emergent themes arise organically from the content of the questions, which can cover a diverse range of topics. The end result of this categorization phase is a collection of thematic categories, as reflected in the questions it comprises. Each theme represents a unique challenge that students are facing in learning to program with the application of counterfactual thinking.

3. Discussions

3.1. Themes

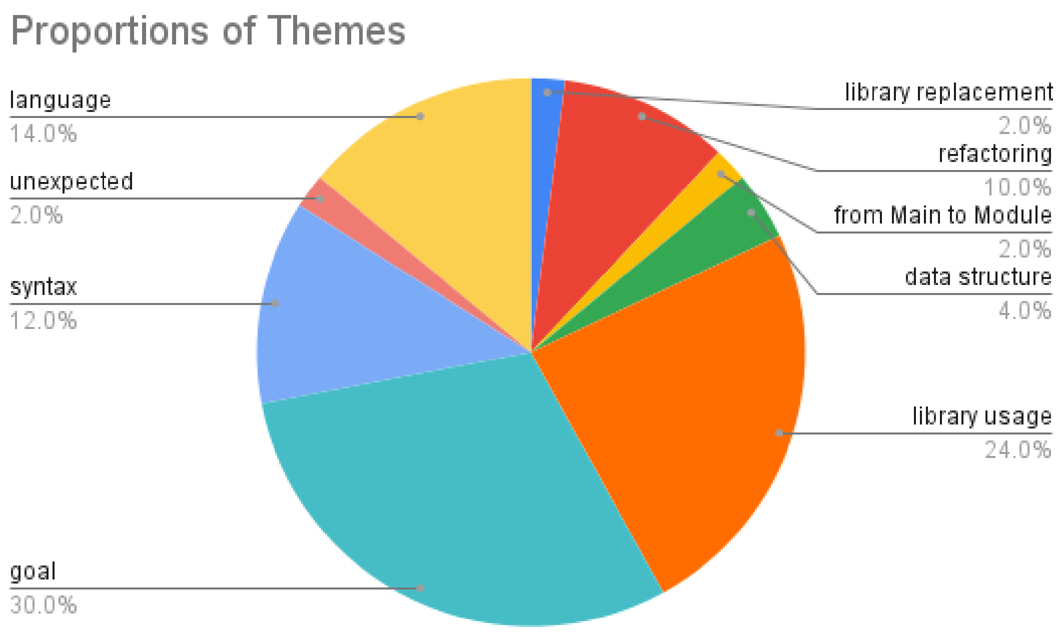

Figure 1: With thematic analysis provides a perspective on the question types

The themes that we found on the generated questions are:

1. **Language Understanding - Syntax**: Pertain to the syntax and the grammatical rules of the programming language. E.g., *"Why is there a semicolon at the end of this line in a Java program?"*.
2. **Language Understanding - Semantic**: Relate to the meaning or purpose of specific programming language keywords or constructs, as well as the principles behind how they operate. E.g., *"What does the 'final' keyword do in Java?"*.
3. **Language Understanding - Other**: Grasping the core concepts, conventions, and idiosyncrasies of a specific programming language. This theme could involve a broad range of language features. E.g., *"Why is the main method necessary in a Java program?"*

4. **Library/Function Understanding**: Understanding how a specific library, module, or function works. This could be a standard library in a language or a third-party library. E.g., *"What does the 'Scanner' class do?"*.
5. **External Behaviour**: How a program interacts with things outside of itself (like input/output operations, network requests, etc.) or how it manifests its results. This could involve its inputs or outputs, how it handles data from a user or a file, or the visible effects of its operation (like changes to the UI or the production of log messages). E.g., *"What will this program print?"* or *"What happens when I click this button?"*.
6. **Refactoring, Internal Behaviour**: How the internal logic of the code could be changed without affecting its external behavior. This could be modified, optimized, or restructured without changing what it does. They can also involve understanding the internal workings of a piece of code, such as the control flow or the interactions between different components of a program. E.g., *"What would happen if we replaced this loop with a map function?"*.

We have summarized our proportions in Figure 1, suggests a richer thematic analysis provides us with sufficient information. This could indicate that the number of codes developed by us may be incomplete, and could be improved.

4. Conclusion & Future Work

While there are several directions to consider with our current work, one potential direction that we are likely going to continue explore is with their use as aids in generating smart content

4.1. Fine-Grained Navigation Support in Intelligent Textbooks

Intelligent textbooks are transforming education [17], offering dynamic and interactive learning experiences. Navigation within these textbooks is a crucial aspect that can enhance learning efficiency and engagement. This study demonstrates how the thematic classification of questions, inspired by counterfactual thinking, and generated by pre–trained Large Language Models (LLMs), can enrich navigational support in intelligent textbooks.

Counterfactual thinking, a mental simulation of alternative realities, often prompted by "what if" scenarios. By combining this psychological construct with the capabilities of pre-trained LLMs, we propose an approach to generate coding-related questions, potentially improving programming proficiency. In our work, even if we do not provide sufficient evidence to the efficacy of the use of LLMs for this task, it indicates small successes in the use of LLMs to generate questions suitable to promote and support program comprehension.

Question themes represent a critical area in computer science education, enhancing learners' problem-solving abilities, coding efficiency, and overall understanding of programming languages. These thematic classifications facilitate learners in personalizing their learning journey and allow for more focused and efficient navigation through the material in intelligent textbooks.

Acknowledgments

Many thanks to Khushboo Thaker and Arezou Farzaneh for their inputs with this work without which this work would not have been complete with.

References

[1] G. Rößling, T. Naps, M. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S. H. Rodger, J. Urquiza-Fuentes, J. �Velázquez-Iturbide, Merging interactive visualizations with hypertextbooks and course management, SIGCSE Bull. 38 (2006) 166–181. doi:10.1145/1189136.1189184.

[2] G. Weber, P. Brusilovsky, ELM-ART: An adaptive versatile system for web-based instruction, International Journal of Artificial Intelligence in Education 12 (2001) 351–384.

[3] E. Fouh, V. Karavirta, D. A. Breakiron, S. Hamouda, S. Hall, T. L. Naps, C. A. Shaffer, Design and architecture of an interactive etextbook – the OpenDSA system, Science of Computer Programming 88 (2014) 22–40. doi:https://doi.org/10.1016/j.scico.2013.11.040.

[4] B. J. Ericson, B. N. Miller, Runestone: A platform for free, on-line, and interactive ebooks, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, Association for Computing Machinery, New York, NY, USA, 2020, p. 1012–1018. URL: https://doi.org/10.1145/3328778.3366950.

[5] Z. Wang, J. Valdez, D. Basu Mallick, R. G. Baraniuk, Towards human-like educational question generation withnbsp;large language models, in: Artificial Intelligence in Education: 23rd International Conference, AIED 2022, Durham, UK, July 27–31, 2022, Proceedings, Part I, Springer-Verlag, Berlin, Heidelberg, 2022, p. 153–166. URL: https://doi.org/10.1007/978-3-031-11644-5_13. doi:10.1007/978-3-031-11644-5_13.

[6] M. B. Manniam Rajagopal, Virtual Teaching Assistant to Support Students' Efforts in Programming, Ph.D. thesis, Virginia Tech, 2018.

[7] S. Barke, M. B. James, N. Polikarpova, Grounded copilot: How programmers interact with code-generating models, Proc. ACM Program. Lang. 7 (2023). URL: https://doi.org/10.1145/3586030. doi:10.1145/3586030.

[8] A. Santos, T. Soares, N. Garrido, T. Lehtinen, Jask: Generation of questions about learners' code in java, in: Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1, 2022, pp. 117–123.

[9] S. Russell, Automated code tracing exercises for cs1, in: Computing Education Practice 2022, 2022, pp. 13–16.

[10] E. Stankov, M. Jovanov, A. Madevska Bogdanova, Smart generation of code tracing questions for assessment in introductory programming, Computer Applications in Engineering Education 31 (2023) 5–25.

[11] A. Henley, J. Ball, B. Klein, A. Rutter, D. Lee, An inquisitive code editor for addressing novice programmers' misconceptions of program behavior, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), IEEE, 2021, pp. 165–170.

[12] L. J. Tamang, R. Banjade, J. Chapagain, V. Rus, Automatic question generation for scaf-

folding self-explanations for code comprehension, in: Artificial Intelligence in Education: 23rd International Conference, AIED 2022, Durham, UK, July 27–31, 2022, Proceedings, Part I, Springer, 2022, pp. 743–748.

[13] T. Lehtinen, L. Haaranen, J. Leinonen, Automated questionnaires about students' javascript programs: Towards gauging novice programming processes, in: Proceedings of the 25th Australasian Computing Education Conference, 2023, pp. 49–58.

[14] C. Schulte, Block model: an educational model of program comprehension as a tool for a scholarly approach to teaching, in: International Computing Education Research Workshop, 2008.

[15] M. T. H. Chi, R. Wylie, The icap framework: Linking cognitive engagement to active learning outcomes, Educational Psychologist 49 (2014) 219 – 243.

[16] A. Webson, A. M. Loo, Q. Yu, E. Pavlick, Are language models worse than humans at following prompts? it's complicated, 2023. arXiv:2301.07085.

[17] P. Brusilovsky, S. Sosnovsky, K. Thaker, The return of intelligent textbooks, AI Magazine 43 (2022) 337–340. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.12061. doi:https://doi.org/10.1002/aaai.12061.

A. Online Resources

All the code and resources are avaiable via Private Github Repository