Use of Large Language Models for Extracting Knowledge Components in CS1 Programming Exercises

Rose Niousha
University of California, Berkeley
Berkeley, USA
rose.n@berkeley.edu

Muntasir Hoq
North Carolina State University
Raleigh, USA
mhoq@ncsu.edu

Bita Akram
North Carolina State University
Raleigh, USA
bakram@ncsu.edu

Narges Norouzi
University of California, Berkeley
Berkeley, USA
norouzi@berkeley.edu

ABSTRACT

This study utilizes large language models to extract foundational programming concepts in programming assignments in a CS1 course. We seek to answer the following research questions: **RQ1.** How effectively can large language models identify knowledge components in a CS1 course from programming assignments? **RQ2.** Can large language models be used to extract program-level knowledge components, and how can the information be used to identify students' misconceptions? Preliminary results demonstrated a high similarity between course-level knowledge components retrieved from a large language model and that of an expert-generated list.

ACM Reference Format:
Rose Niousha, Muntasir Hoq, Bita Akram, and Narges Norouzi. 2024. Use of Large Language Models for Extracting Knowledge Components in CS1 Programming Exercises. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* , 2 pages. https://doi.org/10.1145/3626253.3635592

1 INTRODUCTION AND BACKGROUND

An essential part of designing an inclusive Computing curriculum is to identify essential topics that students will master. Educators create lesson plans and design homework assignments based on the breakdown of topics that must be covered in each course component. The notion of course topics, also known as Knowledge Components (KCs), is introduced within the Knowledge-Learning-Instruction (KLI) framework [3]. KCs represent the competencies students develop through programming exercises, offering profound insights into learning and demonstrating achievement of learning outcomes. The understanding of KCs empowers educators to create more effective CS curricula. However, educators find it challenging to identify which KCs should be part of the curriculum and to what extent KCs are assessed in course assessment elements.

Recent advances in Artificial Intelligence (AI), particularly the emergence of Large Language Models (LLMs), opened up a new avenue in Computing education research. Researchers have used LLMs for generating course materials [6], program repair [4], and IDE-integrated chatbots [5, 8]. LLMs have the potential to facilitate curriculum development for educators and introduce innovative teaching methods. In this work, we explore using LLMs to extract KCs in programming assignments of a CS1 course to facilitate educators' ability to evaluate the coverage of their designed curriculum. We plan to extend the work to extract KCs present in student programs and tie missing expected KCs to misconceptions.

Prior work by Shi et al., KC-Finder, is an automated KC discovery system to uncover KCs within a CS1 course in Java [7]. Shi et al. demonstrated that KC-Finder successfully discovered KC candidates, and students demonstrated improved mastery of the KCs over time. However, the discovered KCs by KC-Finder lacked clearly defined expert concepts and required substantial training data. We address both issues by 1) automatically extracting KCs using LLMs and 2) evaluating the effectiveness of KCs extracted through LLM prompting compared to an expert ground-truth KC list. This study can potentially utilize KC extraction using LLMs to identify student misconceptions and provide personalized feedback.

2 METHODOLOGY AND PRELIMINARY RESULTS

We used a Java programming dataset collected from a CS1 course from the Spring 2019 semester of a public university in the US using the CodeWorkout platform [1]. The course had 50 programming exercises (5 assignments with 10 programming exercises in each). We randomly selected one correct student submission for each problem. We then extracted the problem statements for the corresponding programs. We considered correct submissions since incorrect student programs may lack an associated KC. Next, we prompted two LLMs (Llama2[1] and GPT3.5[2]) with three scenarios:

I. Both Student Program and Problem Description.
II. Only Student Program.
III. Only Problem Description.

We performed prompt engineering and identified the prompt most helpful in extracting KCs. This prompt, in case only the program is passed to LLM, is: *"List ONLY the Knowledge Component*

[1]https://chat.lmsys.org/?arena
[2]https://chat.openai.com/

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0424-6/24/03.
https://doi.org/10.1145/3626253.3635592

name(s) for this code.", and is modified to refer to the description or both description and code in different ablation scenarios.

We aggregated the extracted KCs and removed semantic duplicates to obtain a final list of KCs for the course. Since there were three ablation scenarios and two types of LLMs, the process outlined in Figure 1 is repeated six times, resulting in six KC lists. Two co-authors (CS1 instructors) labeled the dataset with the presented KCs used as the ground-truth data. They first mutually labeled 10% of the data to ensure consistency. Then, they divided the rest of the data and labeled them individually. Due to the cumulative nature of CS1, the importance of KCs dynamically changes over time, and some KCs go into the background (become passive KCs) as they are no longer the point of practice in the new problem. Thus, experts only labeled seemingly active KCs for each problem.

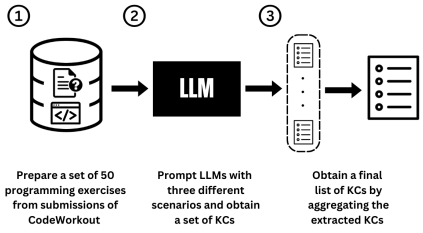

Figure 1: Process of Extracting the List of KCs using LLMs.

2.1 Impact of the Prompt

To assess the level of resemblance between the ground truth and six KC lists, we calculated each Jaccard index.

I. Prompt with student program and problem description. The Jaccardrd index for Llama2 and GPT3.5 were 67% and 83%, respectively.
II. Prompt with only the student program: The Jaccard index for Llama2 and GPT3.5 were 54% and 67%, respectively.
III. Prompt with only problem description: The Jaccard index for Llama2 and GPT3.5 were 28% and 82%, respectively.

Overall, the results show that the KCs generated by LLMs most closely resemble ground truth when prompted with student program and problem description. However, in the third scenario, where GPT3.5 was prompted with only the problem description, the Jaccard index was very close to that of the first scenario. This suggests that GPT3.5 when relying solely on the problem description, can yield responses with a level of similarity comparable to the ideal case, which not only conserves tokens but also holds the potential to reduce the associated cost of automation.

There were minor omissions in the LLMs compared to the ground truth. For instance, the LLMs did not capture the KC "Chain Conditionals" in the ground truth in any scenarios. Moreover, "Nested Loops" were only captured by LLMs in the first scenario, but in the other two scenarios, LLMs only captured "Loops" without specifying the type of loops.

In the third scenario, Llama2 generated unrelated components that were not KCs. Such components include "Lottery Ticket," "Restaurant Table Management," "Temperature and Season,", taken from the scenario described in the question.

2.2 Impact of the LLM

GPT3.5 consistently achieved a higher Jaccard index than Llama2. This suggests that GPT's responses are similar to the ground truth, regardless of the specific prompt scenario. In addition, GPT3.5 had more consistent naming for KCs compared to Llama2.

3 LIMITATIONS

When comparing GPT3.5 and Llama2, one significant difference is that GPT's system stores chat histories, making it easier to continue working from where the last session ended. In contrast, Llama2's system does not have this feature. This difference may explain why the KCs generated by Llama2 struggle with consistent KC naming and sometimes generate unrelated components. Additionally, Our study was limited to a single prompt per problem. We did not investigate potential variations in the outputs of the LLMs that might occur when multiple prompts are applied to the same problem.

4 DISCUSSION AND FUTURE WORK

Our initial findings suggest that LLMs effectively identify course-level KCs that closely match the expert labels. Our next step is to utilize the KCs gathered in this study to evaluate how effectively LLMs can identify KCs within a specific program and how the absence of a KC in a student program connects to misconceptions to provide immediate personalized feedback [2].

Additionally, future work will focus on improving the KC-extraction methodology that is presented in this work. This includes further prompt engineering and identifying the impact of including the potential list of course-level KCs in the prompt to achieve a more consistent and cohesive KC output from LLM. This enables instructors to get responses that use similar terminology as they present in their course when providing student feedback.

5 ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (NSF) under Grants DUE-2236195 and DUE-2331965. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Stephen H Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: short programming exercises with built-in data collection. In *ITiCSE'17*. 188–193.
[2] Muntasir Hoq, Jessica Vandenberg, Bradford Mott, James Lester, Narges Norouzi, and Bita Akram. 2024. Towards Attention-Based Automatic Misconception Identification in Introductory Programming Courses. In *SIGCSE'24 V. 2*.
[3] Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. 2012. The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science* 36, 5 (2012), 757–798.
[4] Charles Koutcheme, Sami Sarsa, Juho Leinonen, Arto Hellas, and Paul Denny. 2023. Automated Program Repair Using Generative Models for Code Infilling. In *AIED'23*. 798–803.
[5] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 2024. Teaching CS50 with AI. (2024).
[6] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *SIGCSE'22 V. 1*. 27–43.
[7] Yang Shi, Robin Schmucker, Min Chi, Tiffany Barnes, and Thomas Price. 2023. KC-Finder: Automated Knowledge Component Discovery for Programming Problems. *International Educational Data Mining Society* (2023).
[8] J.D. Zamfirescu-Pereira, Laryn Qi, Bjoern Hartmann, John DeNero, and Narges Norouzi. 2023. Conversational Programming with LLM-Powered Interactive Support in an Introductory Computer Science Course. In *GAIED @NeurIPS'23*.