Journal of Artificial Intelligence and Technology, 2022, 2, 47-54
https://doi.org/10.37965/jait.2022.0090

ISTP
RESEARCH ARTICLE

AI-infused Semantic Model to Enrich and Expand Programming Question Generation

I-Han Hsiao[1] and Cheng-Yu Chung[2]
[1]Santa Clara University, Santa Clara, CA, USA
[2]Arizona State University, Tempe, AZ, USA

(Received 04 March 2022; Revised 21 March 2022; Accepted 21 March 2022; Published online 31 March 2022)

Abstract: Creating practice questions for programming learning is not easy. It requires the instructor to diligently organize heterogeneous learning resources, that is, conceptual programming concepts and procedural programming rules. Today's programming question generation (PQG) is still largely relying on the demanding creation task performed by the instructors without advanced technological support. In this work, we propose a semantic PQG model that aims to help the instructor generate new programming questions and expand the assessment items. The PQG model is designed to transform conceptual and procedural programming knowledge from textbooks into a semantic network by the Local Knowledge Graph (LKG) and Abstract Syntax Tree (AST). For any given question, the model queries the established network to find related code examples and generates a set of questions by the associated LKG/AST semantic structures. We conduct analysis to compare instructor-made questions from 9 undergraduate introductory programming courses and textbook questions. The results show that the instructor-made questions had much simpler complexity than the textbook ones. The disparity of topic distribution intrigued us to further research the breadth and depth of question quality and also to investigate the complexity of the questions in relation to the student performances. Finally, we report a user study results on the proposed Artificial Intelligent-infused semantic PQG model in examining the machine-generated questions' quality.

Key words: Assessment; Programming; Semantic Modeling; Automatic Question Generation

I. INTRODUCTION

Questions are one of the most widely used tools to assess students' knowledge acquisition [12]. Well-engineered questions permit students to assess the critical aspects of the subject, provide a bridge between theory and practice from the question and the answer, and offer useful problem-solving applications. Numerous studies have demonstrated the effectiveness of questions utilization via various educational technologies [3]. For instance, instructors can periodically release questions (e.g., quizzes, exercises, worked examples) on a self-assessment platform and help students regularly and persistently practice the content [7,8]. With the rise of the remote and distance learning, the role of questions is becoming more imperative than ever. Although ranges of educational technologies have made progress in harnessing reusable educational objects and depositories [2,4,5,6,14,16] or crowdsourcing-based method of generating questions [10], the proposed Artificial Intelligent (AI)-based approaches predominantly focus on language translation methods; for example, [26], less is emphasized on AI-based semantic approaches to automatically generate meaningful and engaging questions. Therefore, our research group explores AI-infused semantic modeling in assisting the *human-intelligence-demanding* process [15].

In our targeted domain, programming language learning, it has been pointed out that making a programming question is not as simple as transforming declarative sentences into an interrogative format. One main challenge is the alignment between *knowing-that* (conceptual) and *knowing-how* (procedural) knowledge. The instructor not only needs to consider students' abilities but also carefully designs questions to target desired skills by integrating programming code and concepts. These aspects indicate that the content of the questions usually transcends beyond the textual representation, which raises questions like "*what are appropriate question structures or complexity?* " and "*how are they related to student performance?"* We have previously examined the AI + HCI approach to facilitate instructor in making exam questions [13]. In this work, we focus on engineering a semantic programming question generation (PQG) model to facilitate question creation.

We adopt a semantic network model, Local Knowledge Graph (LKG) to represent a question in the format of semantic triples (i.e., combinations of verb, subject, and object) and to construct a semantic network around them. We hypothesize that programming code and its intents can be synthesized by their descriptions in verb-arguments formats, thereby building a network in an unsupervised way by the Abstract Syntax Tree (AST) and the LKG model [21]. The PQG model can be used to make programming questions by AST nodes and the associated semantic triplets (i.e., *subject-verb-object* pairs). We believe the model can help instructors make programming questions more efficient. Overall, the proposed work is guided by the following research questions:

1. How do we augment conceptual and procedural knowledge in computer programming in a PQG model?
2. To what extent the semantic PQG model affect the generated question quality?

We conduct analysis to compare instructor-made questions from 9 undergraduate introductory programming courses and textbook

Corresponding author: I-Han Hsiao (e-mail: ihsiao@scu.edu).

questions. The results show that the instructor-made questions had much simpler complexity than the textbook ones. The disparity of topic distribution intrigued us to further research the breadth and depth of question quality and also to investigate the complexity of the questions in relation to the student performances. Therefore, in this paper, we report the methodology of the proposed AI-infused semantic PQG model and a user study to uncover valuable insights of the PQG extensibility.

II. RELATED WORK

A. AUTOMATIC QUESTION GENERATION AND ANALYSIS

The two fundamental concepts of QG are "*what to ask*" and "*how to ask.*" A QG process works like a processing pipeline where the key information from the input ("*what to ask*") is extracted and transformed into question forms ("*how to ask*"). A recent review of QG for educational purposes from Kurdi et al. [15] categorizes QG methods into two aspects: understanding and transformation.

The level of understanding describes how a QG method extracts key information in a given context. There are two common types of extraction in the literature: syntax-based extraction and semantic-based extraction. A syntax-based method uses syntactic features from input, for example, part-of-speech tags, named entity recognition (NER), and parse-tree dependency. Such features represent structural characteristics of the input and to some extent the meaning. For example, Willis et al. [24] analyzed the Stanford Question Answering Dataset (SQuAD) [19] that consisted of context-question-answer pairs sourced from Wikipedia. They found the five most frequent NER tags in the questions were MONEY, CARDINAL, PERCENT, DATE, and PERSON. This result suggested that key information (i.e., key phrases in text) was highly correlated with phrases that have certain syntactic features.

The other kind of extraction is semantic-based and requires the understanding of input beyond the syntactic features. One prevalent design of semantic-based methods is using the knowledge base (ontology) that provides a structured knowledge representation. A knowledge base consists of key concepts and their dependency upon each other. For example, Zhang & VanLehn [26] proposed a QG method for introductory biology classes. They obtained an existing knowledge base about biology and defined a set of question schema for different semantic triples (i.e., data entities in the format of subject-predicate-object expressions). Compared to syntax-based methods, a semantic-based method may generate questions that are more relevant to the target domain because noise has been filtered out during the construction of knowledge bases.

B. PROGRAMMING QUESTION GENERATION FOR EDUCATIONAL PURPOSES

QG for educational purposes has been explored in different fields of study, including mathematics, physics, biology, and linguistics. A recent survey in the literature has shown that automatic QG is especially popular in the field of language learning [15]. The learning content has a close relationship to the grammar, syntax, and semantics of natural languages, which makes the field an obvious subject of QG. In addition, the need for standardized exams also engages more research to focus on the field [15]. In addition to formal learning, much research on QG has also focused on the development of general QA systems for training and instructions, for example, [20].

Programming learning nowadays has been reshaped by the use of educational technology that emphasizes personalized learning [9]. For example, Lu & Hsiao [20] uses adaptive learning techniques to personalize the search result for programming learners. For another example, [1] focuses on bite-size practices that distribute student practice over time to achieve the distributed learning effect. These educational technologies are all based on the abundant supply of high-quality materials that can not only fulfill the quantity in need but also the quality of personalized learning which requires a sufficient coverage of concepts.

However, despite the need for more questions and the popularity of programming learning, little research has been focused on QG for computer programming learning. An early work from [4] focuses on individualized, parameterized exercises for the C programming language. Similar to mathematical formulas, coding questions in programming learning can be transformed into fixed templates with part of them replaced by variables. The variables can then be replaced by predefined cases (i.e., values in worked examples) to generate a large number of new questions. Another research from [27] focuses on the SQL programming language which has a structure similar to natural languages. They propose a deep neural network for translating the natural language content to corresponding SQL queries for questions. Overall, we believe there is still a need for further research in the QG for programming learning.

III. METHODOLOGY

A. MODELING CONCEPTUAL KNOWLEDGE BY OPEN INFORMATION EXTRACTION AND LOCAL KNOWLEDGE NETWORK

Semantic role labeling (SRL) is the process that identifies semantic relationships between words or phrases in sentences. SRL is widely used in natural language processing to build a model that can interpret the syntactic structure of natural language. It also helps index textual information by compact formats like the "*(subject)-verb-objects*" relationship which is usually stored as triples, for example, (a program/ARG; stored/VERB; objects/ARG) (Fig. 1).

The main task of automatic SRL systems is to group a sequence of words into a set of predefined SRL labels. However, labeling words by the rules of grammar may not be sufficient for the machine to understand natural language. For example, a complex sentence may have an overlapping structure and include more than one proposition. The sentence can therefore be decomposed into different possible SRL triples, each of which reveals different aspects of the meaning. Extending the concept of SRL, researchers have proposed Open Information Extraction (OIE) that considers both SRL and propositions asserted by sentences [25]. For example, the sentence "computers connected to the Internet can communicate with each other" can be decomposed into two possible propositions: *"(Computers connected to the Internet; can; communicate with each other)*" and *"(Computers; connected; to the Internet).*" These two propositions represent two aspects of the meaning. Compared to conventional SRL, OIE can extract more information about the intent of a given sentence.

Based on OIE, the LKG is a semantic network model that connects subjects, verbs, and objects in OIE triples [11]. It has been shown that the LKG can be used to store a large volume of documents and provide an efficient structure for search queries. We believe that the LKG can also help analyze the structure of question content due to its network structure: the interconnection of similar semantic objects and the span of edges may represent the

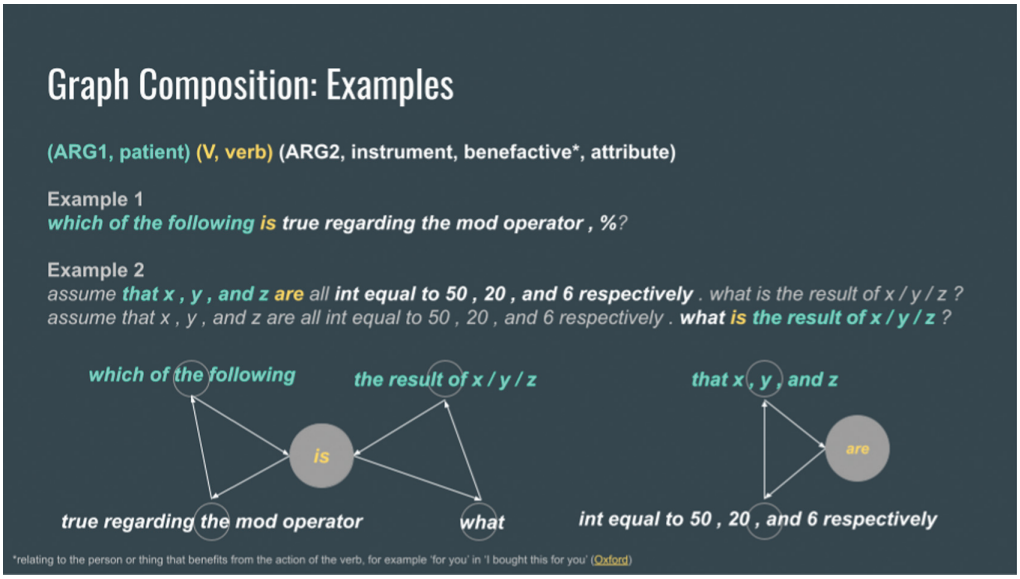

Fig. 1. Examples of LKG in our context.

complexity of underlying knowledge, thereby forming a semantic network. Once questions are represented as OIE triples in an LKG, we can further analyze the complexity of questions in terms of common statistics in the field of network analysis [22]. This enables the opportunity to understand not only the difference between instructor-made questions and textbook questions but also the correlation between the complexity of questions and student performance.

B. MODELING PROCEDURAL PROGRAMMING KNOWLEDGE BY ABSTRACT SYNTAX TREE

A programming language is usually defined by a formal language with well-structured grammar. This characteristic ensures that program code can be efficiently parsed into binary machine code by a compiler. The AST is an alternative representation of programs that specifically focuses on the syntactic structure. For example, in an AST of Java code, the node "*ClassOrInterfaceDeclaration*" represents the entry point of a Java class definition, and the node "*VariableDeclarator*" represents a statement that declares a new variable and its initializer. Although the AST is not necessarily related to the runtime nature of programs (i.e., the references to external libraries or the actual flow of data), it provides a convenient way to parse and represent programming semantics.

C. EXTRACTION OF SEMANTIC TRIPLES AND CONSTRUCTION OF LKG

This study adopts a supervised OIE model developed by [21] to automatically extract propositions from questions. The model is based on an RNN architecture trained to iteratively extract propositions from a given sequence of words. The output of OIE is a set of triples that represent the propositions in a given question. Each triple always consists of one verb (VERB) and the associated arguments (ARG) with index numbers that suggest their semantic roles. The labeling schema follows the PropBank corpus [18] where ARG0 denotes the agent (subject) of the verb, ARG1 denotes the patient (direct object) of the verb, and ARG2 denotes the instrument (or the entity given to ARG1). Higher orders of arguments (e.g., ARG3, ARG4, etc.) and modifiers were omitted in the analysis due to their rarity in the dataset. After triples were extracted by the OIE model, we constructed an LKG in a similar manner to the work from [11] where the researchers used subjects/objects as nodes and verbs as edges. The only difference was that instead of using verbs as edges, we made verbs also nodes of the network and built edges by the subject-verb-object relationship.

D. DATA COLLECTION

We collected three datasets of questions, one of which represent instructor-made questions and the other textbook questions. The first dataset, QuizIT, was instructor-made multiple-choice questions (MCQ) collected from [1] that was used in 9 undergraduate courses about entry-level Java programming over 3 years. The instructor might make questions from scratch or select/modify the existing questions in a question bank. Because an MCQ from QuizIT might have the major content of questions in the options (e.g., "*Which of the following is correct?*"), we chose to concatenate all question texts with their answer options to ensure that sufficient details were included in the dataset. The number of unique questions was 779 (355 when counting unique question text only). The second dataset of practice questions, Textbook, was collected from a free online textbook, "Introduction to Programming Using Java, Eighth Edition."[1] We collected 7 chapters of content that overlapped with the 9 undergraduate courses on Introduction to Programming. In total, there were 163 practice questions, most of which were free-text questions. The third dataset, QBank, was collected from a question bank accompanied by the textbook used in the entry-level Java programming course. After processing, we collected 225 practice questions that were also in the format of MCQ. The student performance was represented by the statistical first-attempt error rate on QuizIT. This dataset consisted of records from 570 students who contributed 14,534 first attempts. More details about this performance data are described in the analysis below.

[1] https://math.hws.edu/javanotes/

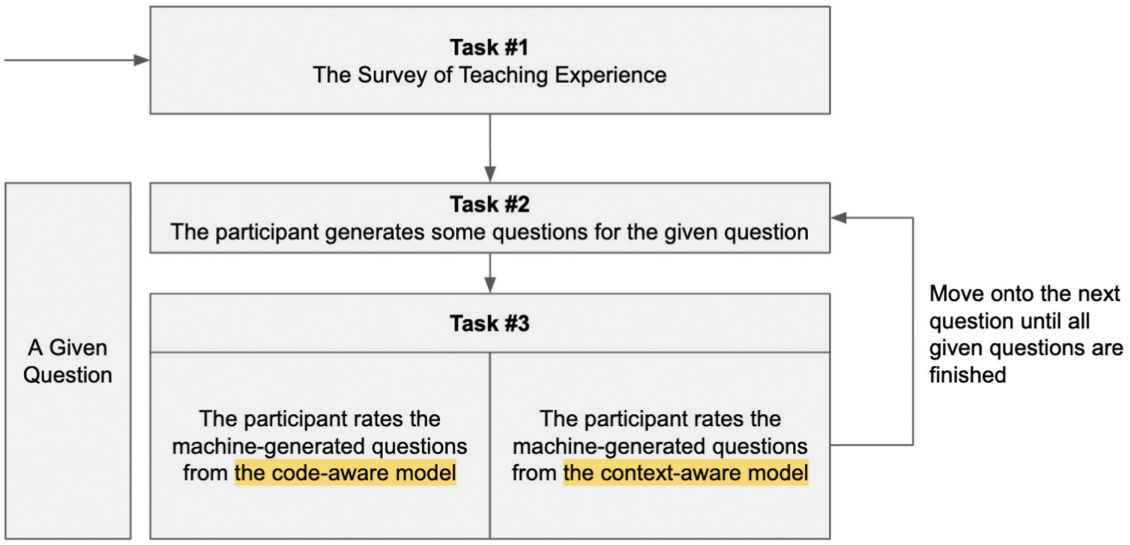

Fig. 2. User study procedure.

E. STUDY DESIGN

To evaluate the performance of the PQG model, we designed a user study (Fig. 2) that aimed to collect feedback from instructors who have experience in teaching introductory programming courses. The user study consists of two parts: a survey of teaching experience and a task of question evaluation. A participant is expected to spend around 1 hour to finish the whole study.

In the evaluation task, the participant is given 12 sets of input questions and generated questions. The input questions were selected from a pool of difficult questions according to their statistical difficulty (i.e., the error rate) on a self-assessment platform [1]. Considering the cost of the user study, especially the recruitment of experienced instructors, we limited the number of input sets to 12.

For each input question, s/he has to generate at least one new programming question and evaluate the quality of the generated questions according to (1) the relevance to the topic of the input (Topic-Rel), (2) the extensibility in terms of topics (Ext-Topics), (3) the extensibility in terms of complexity (Ext-Complex), and (4) the extensibility in terms of the participant's need of question generation (Ext-Need). All of the evaluation questions are 5-item Likert scales that ranged from -2 to $+2$.

As far as we know, there is no existing and publicly available model or benchmark datasets of PQG. To compare the performance of the model with a reference, we devised a reference model by masking part of the proposed PQG model. The reference model, called the "*code-aware*" model, uses only the AST structures to generate programming questions. The reference model is compared to the other model called the "*context-aware*" model which uses the LKG structure to generate programming questions. To some extent, the result of this comparison can indicate the performance of the knowledge representation and whether the participants have any preference for the model.

IV. EVALUATION

A. SEMANTIC PQG MODEL QUALITY

1) TEXTBOOK QUESTIONS HAVE A HIGHER DIVERSITY AND COMPLEXITY THAN INSTRUCTOR-MADE QUESTIONS. All the networks had many connected components on the circumference that was separate from the major component in the center (as shown in Fig. 1). In our construction of the LKGs, propositions using the same verbs or arguments would reside in the same connected component. A component that was separate from the others suggested that the underlying propositions used a different set of verbs/arguments. Following this interpretation, the number of connected components became an indication of the diversity of concepts because different concepts might be addressed by different verbs/arguments, therefore resulting in different formats of questions. The largest number of connected components was QBank with 113 components. The Textbook had 74, and QuizIT had 53. This result suggests that questions of QBank might have a higher diversity of concepts than the others. In contrast to this, QuizIT, even though it had the greatest number of questions, resulted in the lowest diversity of concepts. This could be due to the smaller vocabulary used by the instructors when they addressed concepts or that the instructors tended to make questions in similar and simple formats.

2) QUESTION COMPLEXITY IS A SIGNIFICANT PREDICTOR OF STUDENT PERFORMANCE. We conducted a classification analysis to evaluate the classification performance of the combination of unique connected components (CUCC) and other network characteristics. The Random Forest classifier with 10-fold cross-validation was used to benchmark different configurations of features. The result showed that a model using one-hot vectors of the CUCC alone was able to retrieve most high-error questions ($f1 = 0.67$, $precision = 0.54$, $recall = 0.90$, $accuracy = 0.58$); however, its precision was mediocre. After several iterations of experiments, we found two network characteristics that helped reach the best performance: the out-degree of verbs (ODV) and the betweenness of argument (BTA). The classification reached $f1 = 0.72$, $precision = 0.75$, $recall = 0.70$, and $accuracy = 0.73$. Although the recall was lower than the classification with the CUCC only, these features seemed to find a reasonable balance. This result suggests that the ODV and the BTA might be indirect indicators of the CUCC. Hypothetically, a small component had low ODV and high BTA; a large component had high ODV and low BTA. Although the current state of analysis was not able to examine this hypothesis, the result from the classification analysis

can help develop a measure to evaluate the complexity of questions.

B. SEMANTIC PQG MODEL IMPACT

1) HIGH SIMILARITY BETWEEN THE INSTRUCTOR-GENERATED QUESTIONS AND MACHINE-GENERATED QUESTIONS. To find out the semantic similarity between the collected instructor-generated questions and the corresponding machine-generated questions, we trained a Word2Vec embedding out of public programming textbooks. The embedding model was then used to transform the input questions (*IN*), the instructor-generated questions (*USER*), and machine-generated questions (*CA* for code-aware; *CT* for context-aware) into vectors by which we were able to compute the cosine similarity (*SIM*) between them as shown in Fig. 3.

The result showed that the SIM(CA, USER) (M = 0.84, SD = 0.14) and the SIM(CT, USER) (M = 0.82, SD = 0.19) were similar and significantly higher than the SIM(IN, USER) (M = 0.63, SD = 0.25) ($t(166) = 6.72$, $p = 0.00$; $t(166) = 5.43$, $p = 0.00$). The relatively low SIM(IN, USER) to some extent suggests that the instructor-generated questions addressed concepts that were more extensive than the input questions, which is expected because the instructors were asked to generate new questions that can help their students to practice the concept from the input question. The significantly high and similar SIM(CA, USER) and SIM(CT, USER) suggest that the machine-generated questions addressed concepts similar to the instructor-generated questions. Also, there was no significant difference between the code-aware and the context-aware models. These outcomes plausibly indicate that the knowledge extraction of our PQG model was aligned with the instructors' opinions. In other words, the model was able to generate questions that were similar to those generated by the instructor.

2) SIGNIFICANTLY POSITIVE RATINGS ON THE UTILITY OF MACHINE-GENERATED QUESTIONS. We summarized in Fig. 4 and Table I the instructors' subjective opinions on the machine-generated questions over the topic relevance and the extensibility in terms of topics, complexity, and instructional needs. First, we found that both experiment models received a significantly positive (with respect to zero) rating, especially on the extensibility of complexity. There was no significant difference found between the two models. This result suggests that the experienced instructors were generally satisfied with the utility of the machine-generated questions. They found the machine-generated questions relevant to the input questions. The machine-generated questions also provided sufficient details for them to generate new questions that address similar topics and are complex enough to distinguish high-performing and low-performing students. Overall, our PQG model was able to supply sufficient information for them to generate new questions for students.

Interestingly, the code-aware model did not perform worse than the context-aware model, which rejects our hypothesis about their performance (the code-aware model covers a narrower range of knowledge than the context-aware model, therefore being an inferior option). Plausibly, this result suggests that our PQG model was helpful for the instructors to generate new questions no matter if it was only code-aware or only context-aware, even though the effect of the full model remains unclear and requires further analysis. Due to the similar performance of the two masked models,

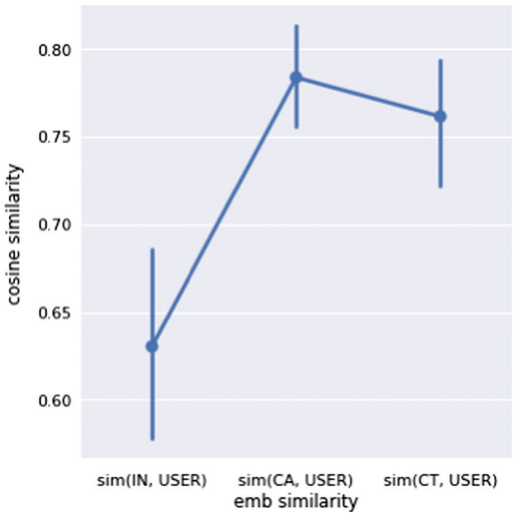

Fig. 3. Embedding similarity between the input (IN), machine-generated (CA = code-aware, CT = context-aware), and instructor-generated (USER) Questions.

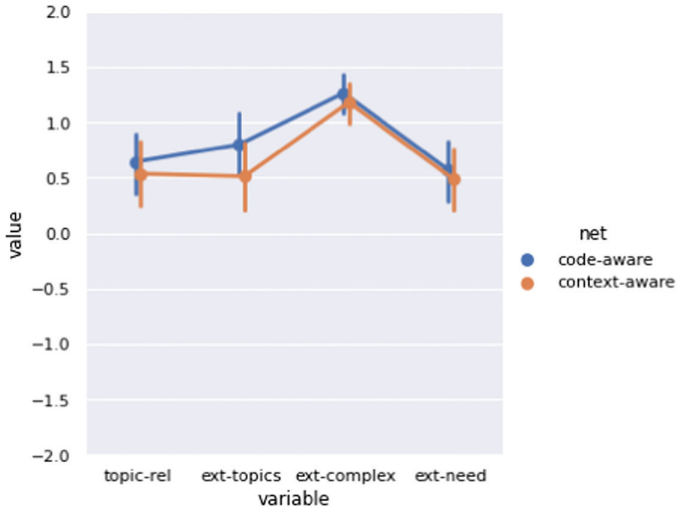

Fig. 4. The distribution of ratings to the four utility variables: topic relevance (Topic-Rel), extensibility of topics (Ext-Topics), complexity (Ext-Complex), and the instructor's need (Ext-Need). The instructors consistently gave significantly positive ratings to all the variables.

Table I. The statistics of one-sample T-tests for the four rating variables (Reported in the Format "M, SD, test (DoF) = V (P VAL)"

	Code-aware	Context-aware
Topic-Rel	0.64, 1.34, $t(83) = 4.39$ (0.00)	0.64, 1.34, $t(83) = 3.59$ (0.00)
Ext-Topics	0.80, 1.27, $t(83) = 5.76$ (0.00)	0.80, 1.27, $t(83) = 3.37$ (0.00)
Ext-Complex	1.26, 0.81, $t(83) = 14.31$ (0.00)	1.26, 0.81, $t(83) = 12.67$ (0.00)
Ext-Needs	0.57, 1.24, $t(83) = 4.21$ (0.00)	0.57, 1.24, $t(83) = 3.62$ (0.00)

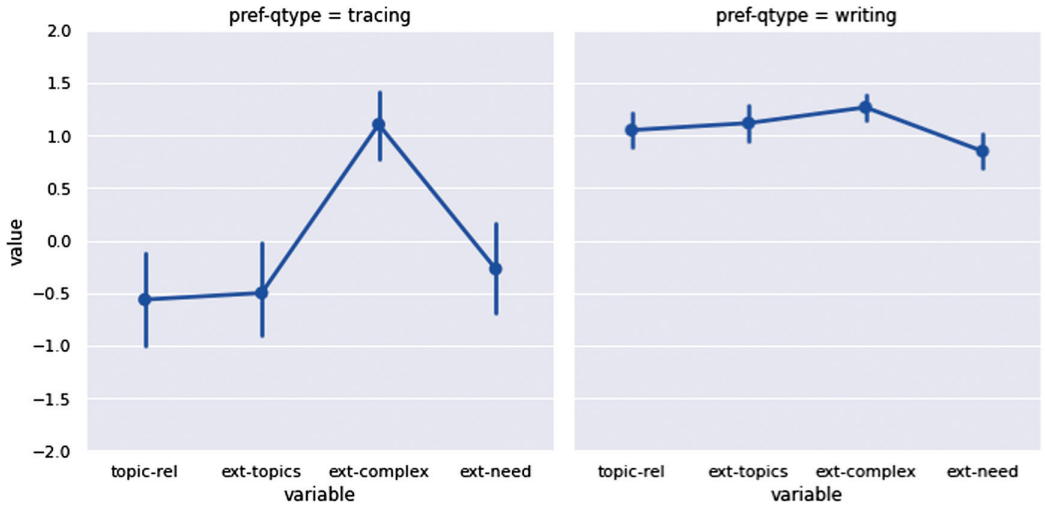

Fig. 5. Multilevel analysis of ratings and preferred question types. The instructors who prefer code-writing questions gave consistently positive ratings to all variables; however, those who prefer code-tracing questions gave mixed ratings except for the extensibility of complexity.

the following analyses report the ratings with the two combined for readability.

3) THE PQG MODEL HELPS GENERATE GENERAL CODE-WRITING QUESTIONS AND COMPLEX CODE-TRACING QUESTIONS. It is likely that instructors need different kinds of support when generating new programming questions. To find out whether our PQG model can address certain needs of the instructors, we conducted a multilevel analysis by factoring in the preferred question types as shown in Fig. 5 and Table II. The analysis showed that the instructors who valued code-writing questions were the major source of positive ratings in the evaluation. They unanimously expressed significantly positive ratings on all four variables. However, the instructors who valued code-tracing questions the most had mixed ratings: Only the extensibility of complexity received a significantly positive rating from this group.

This result is interesting as it points out that our PQG model may only address the needs of a certain population. For code-writing questions, the PQG model was able to provide topic-relevant and highly extensible questions for the instructors to use after edits in practice. However, for code-tracing questions, the model was limited and only able to provide questions that were complex enough but not topic-relevant nor extensible in terms of topics and instructional needs. One plausible explanation is that the instructors might expect code-tracing questions to be more aligned with the content they use in class, for example, small code snippets or worked examples in the instructions. Nevertheless, the design of our PQG model is based on the content of public textbooks that may not include such an example. Considering the target audience of the instructors is students from introductory programming courses, it is likely that our PQG model generated questions that were too complex for the students to learn, which is partially confirmed by the high rating for the variable Ext-Complex.

Table II. The one-sample statistics of the multilevel analysis (Preferred question types and ratings)

Preferred Question Type	Variable	Statistics
Code-tracing	Topic-Rel	−0.56, 1.57, t(47) = −2.46, 0.02
	Ext-Topics	−0.50, 1.49, t(47) = −2.31, 0.03
	Ext-Complex	1.10, 1.07, t(47) = 7.11, 0.00
	Ext-Needs	−0.27, 1.54, t(47) = −1.21, 0.23
Code-writing	Topic-Rel	1.05, 0.90, t(119) = 12.69, 0.00
	Ext-Topics	1.12, 0.92, t(119) = 13.19, 0.00
	Ext-Complex	1.27, 0.70, t(119) = 19.63, 0.00
	Ext-Needs	0.85, 0.91, t(119) = 10.19, 0.00

V. CONCLUSIONS

A. SUMMARY

A PQG model that aims to support instructors to make new programming questions from the existing ones. Following the knowledge-based QG approach, we used the LKG to represent the conceptual programming knowledge and the AST to represent procedural programming knowledge. We conducted a user study with experienced instructors from introductory programming courses. The results showed that the participants had significantly positive feedback toward the extensibility of question complexity. Overall, this work contributes to the understanding of instructors' question generation process by an explainable data model and paves a road to the future development of AI-assisted PQG tools for educational purposes.

B. LIMITATIONS & FUTURE WORK

There are several limitations of this study. First of all, the evaluation task covered a relatively small scope and a limited number of topics in programming learning. Programming topics may range from basic syntax to advanced knowledge. The evaluation included also a small set of machine-generated questions. Further research is still required to validate the model's capability in different topics. Second, the influence on the rating from part of the machine-generated content remains unclear. The current PQG model generates both code examples and questions about the examples.

However, the evaluation task did not ask the instructor to evaluate them individually or separately. The instructors might give more weight to either the examples or the questions in their ratings. Third, the sample size in this study was relatively small due to the high cost of recruiting experienced instructors. Although the feedback from such a population is valuable, it still requires a larger sample to generalize the findings of this study.

Finally, due to the lack of widely available datasets for the performance analysis of PQG models, this study used two masked models and compared their performance to each other and a set of instructor-generated questions. This made the analysis relatively subjective with respect to the recruited instructors' preferences. Future studies should consider using the result of this study as one baseline reference or developing a more rigorous design for PQG performance analysis.

REFERENCES

[1] M. Alzaid and S. Hsiao, "Utilising problem-solving: from self-assessment to self regulating," *New Rev. Hypermedia Multimed.*, vol. 25, no. 3, pp. 222–244, 2019.

[2] M. Baker, X. Hu, G. De Luca, and Y. Chen, "Intelligent voice instructor-assistant system for collaborative and interactive classes," *J. Artif. Intell. Technol.*, vol. 1, no. 2, pp. 121–130, 2021. DOI: 10.37965/jait.2021.0003.

[3] R. S. J. d. Baker, A. T. Corbett, and V. Aleven, "More accurate student modeling through contextual estimation of slip and guess probabilities in Bayesian knowledge tracing," in *Intelligent Tutoring Systems*, vol. 5091 LNCS. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 406–415. DOI: 10.1007/978-3-540-69132-744. Available: http://link.springer.com/10.1007/978-3-540-69132-744.

[4] P. Brusilovsky and S. Sosnovsky, "Invidualized exercises for self-assessment of programming knowledge: an evaluation of QuizPACK," *J. Educ. Resour. Comput.*, vol. 5, no. 3, p. 6, 2005. DOI: 10.1145/1163405.1163411.

[5] P. Brusilovsky, M. Yudelson, and I. H. Hsiao, "Problem solving examples as first class objects in educational digital libraries: three obstacles to overcome," *J. Educ. Multimed. Hypermed.*, vol. 18, no. 3, 267–288, 2009.

[6] R. Cafolla, "Project MERLOT: bringing peer review to web-based educational resources," *J. Technol. Teacher Educ.*, vol. 14, no. 2, pp. 313–323, 2006.

[7] C. Y. Chung, and I. H. Hsiao, "Investigating patterns of study persistence on self-assessment platform of programming problem-solving," in *Proc. 51st ACM Technical Symposium on Computer Science Education*, Feb. 2020, pp. 162–168.

[8] C. Y. Chung and I. H. Hsiao, "From detail to context: modeling distributed practice intensity and timing by multi-resolution signal analysis," *Int. Educ. Data Mining Soc.*, 2021.

[9] B. C. Czerkawski and E. W. Lyman, "Exploring issues about computational thinking in higher education," *TechTrends*, vol. 59, no. 2, pp. 57–65, 2015. DOI: 10.1007/s11528-015-0840-3.

[10] P. Denny, A. Luxton-Reilly, and J. Hamer, "The PeerWise system of student contributed assessment questions," in *Proc. Tenth Conference on Australasian Computing Education-Volume 78*, Jan. 2008, pp. 69–74.

[11] A. Fan, C. Gardent, C. Braud, and A. Bordes, "Using local knowledge graph construction to scale Seq2seq models to multi-document inputs," in *EMNLP-IJCNLP 2019-2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, 2019, pp. 4186–4196. DOI: 10.18653/v1/d19-1428.

[12] Y. Ham and B. Myers, "Supporting guided inquiry with cooperative learning in computer organization," in *Proc. 50th ACM Technical Symposium on Computer Science Education*, Feb. 2019, pp. 273–279.

[13] I. H. Hsiao, P. Brusilovsky, and S. Sosnovsky, "Web-based parameterized questions for object-oriented programming," in E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Association for the Advancement of Computing in Education (AACE), Nov. 2008, pp. 3728–3735.

[14] K. R. Koedinger, R. S. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper, "A data repository for the EDM community: the PSLC DataShop," *Handb. Educ. Data Mining*, vol. 43, pp. 43–56, 2010.

[15] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, "A systematic review of automatic question generation for educational purposes," *Int. J. Artif. Intell. Educ.*, vol. 30, no. 1, pp. 121–204, 2020. DOI: 10.1007/s40593-019-00186-y.

[16] M. Lovett, O. Meyer, and C. Thille, "The open learning initiative: measuring the effectiveness of the OLI statistics course in accelerating student learning," *J. Interact. Media Educ.*, 2008.

[17] Y. Lu and I.-H. Hsiao, "Personalized information seeking assistant (PiSA): from programming information seeking to learning," *Inf. Retr. J.*, vol. 20, no. 5, pp. 433–455, 2017. DOI: 10.1007/s10791-017-9305-y.

[18] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: an annotated corpus of semantic roles," *Comput. Linguist.*, vol. 31, no. 1, pp. 71–106, 2005. DOI: 10.1162/0891201053630264

[19] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: unanswerable questions for SQuAD," arXiv, 2018. Available: http://arxiv.org/abs/1806.03822.

[20] S. Ruan, L. Jiang, J. Xu, B. J.-K. Tham, Z. Qiu, Y. Zhu, E. L. Murnane, E. Brunskill, and J. A. Landay, "QuizBot: a dialogue-based adaptive learning system for factual knowledge," in Proc. 2019 CHI Conference on Human Factors in Computing Systems, Chi, 2019, pp. 1–13. DOI: 10.1145/3290605.3300587.

[21] G. Stanovsky, J. Michael, L. Zettlemoyer, and I. Dagan, "Supervised open information extraction," in NAACL HLT 2018–2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies – Proceedings of the Conference 1(Section 4), 2018, pp. 885–895. DOI: 10.18653/v1/n18-1081.

[22] A. Veremyev, A. Semenov, E.L. Pasiliao, and V. Boginski, "Graph-based exploration and clustering analysis of semantic spaces," *Appl. Netw. Sci.*, vol. 4, no. 1, p. 104, 2019. DOI: 10.1007/s41109-019-0228-y. Available: https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0228-y.

[23] T. Wang, X. Yuan, and A. Trischler, "A joint model for question answering and question generation," arXiv preprint arXiv: 1706.01450, 2017.

[24] A. Willis, G. Davis, S. Ruan, L. Manoharan, J. Landay, and E. Brunskill, "Key phrase extraction for generating educational question-answer Pairs," in *Proc. Sixth (2019) ACM Conference on Learning @ Scale*, New York, NY, USA, ACM, Jun. 2019, pp. 1–10. DOI: 10.1145/3330430.3333636. Available: https://dl.acm.org/doi/10.1145/3330430.3333636.

[25] A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland, "TextRunner," in *Proc. Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations on XX – NAACL '07*, Morristown, NJ, USA, Association for

Computational Linguistics, 2007, pp. 25–26. DOI: 10.3115/1614164.1614177. Available: http://portal.acm.org/citation.cfm?doid=1614164.1614177.

[26] L. Zhang and K. VanLehn, "How do machine-generated questions compare to human-generated questions?," *Res. Pract. Technol. Enhanc. Learn.*, vol. 11, no. 1, 2016. DOI: 10.1186/s41039-016-0031-7.

[27] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: generating structured queries from natural language using reinforcement learning," ArXiv, pp. 1–12, 2017. http://arxiv.org/abs/1709.00103.