Automatic Question Generation for Scaffolding Self-Explanations for Code Comprehension *

Lasang J. Tamang, Rabin Banjade, Jeevan Chapagain, and Vasile Rus

University of Memphis, Memphis, TN, USA
{ljtamang,rbnjade1,jchpgain,vrus}@memphis.edu

Abstract. This work presents two systems, Machine Noun Question Generation (QG) and Machine Verb QG, developed to generate short questions and gap-fill questions, which Intelligent Tutoring Systems then use to guide students' self-explanations during code comprehension. We evaluate our system by comparing the quality of questions generated by the system against human expert-generated questions. Our result shows that these systems performed similarly to humans in most criteria. Among the machines, we find that Machine Noun QG performed better.

Keywords: Automatic question generation · Self-explanation · Program comprehension · Intelligent Tutoring System · Authoring

1 Introduction

This work is part of our larger effort to develop Intelligent Tutoring Systems (ITS) to help students learn computer programming. Such ITS uses questions to be provided as hints meant to scaffold students' self-explanation [9] during code comprehension. When done by human experts, which is currently the norm, authoring such questions is expensive and hard to scale, often taking 100-200 hours to prepare 1-hour instructional content [1]. In this work, we develop two systems called Machine Noun Question Generator (QG) and Machine Verb QG for automatically generating short questions and gap-fill questions using expert generated code-block explanations that ITS employs to scaffold student self-explanation during code comprehension.

Some prior works [10, 11, 5] automatically create clones of programming exercises that provide opportunities to practice more as opposed to scaffolding students' self-explanation for the particular code, which is the focus of our work. Other works like [2, 8] automatically generated short questions from static analyses of code, using the template-based QG approach, which requires significant time to design the templates. Unlike past work, we do not use a template approach for question generation. Instead, we use the current state-of-art model ProphetNet [6] which inputs textual explanations of the code, leading to a more

* This work is supported by the National Science Foundation under grant number 1822816 and 1934745. All findings and opinions expressed are solely the authors'.

computer language-independent approach for question generation. Also, it can produce a more profound and broader variety of questions compared to the limited type of questions that the expert-provided templates can generate.

In sum, this paper answers the following research questions:

1. Is it possible to automatically generate short questions that are linguistically well-formed, pedagogically sound, and indistinguishable from human-generated questions?
2. Is it possible to automatically produce gap-fill questions useful for ITS?
3. How do questions generated by machines compare to expert questions?
4. How do Machine Noun QG and Machine Verb QG compare in teperformance?

2 Dataset

The dataset used for this work consists of 10 code examples with explanations followed by short and gap-fill questions for each code block, as shown below, prepared and refined by our group of subject experts in several iterations.

```
public class AverageOfNumbers {
 public static void main(String[] args) {

    /* Code-Block 1, Expert-Explanation, short and gap-fill questions*/
    double[] numArray = {8,6,11,7};
    double sum = 0.0; double average;

    /* Code-Block 2
    The sum of numbers is calculated using a for loop that iterates over
    each number in the numArray array and adds each number to the sum.
    When the for loop completes execution, the value of the sum is 32.
    1. How is a sum of numbers calculated?
    The sum of numbers is calculated using a _____ that iterates over
    each number of numArray and adds each number to the sum.
    2. What is the value of sum when the for loop completes execution?
    When the forloop completes execution, the value of the sum is __.
    */
    for (int i = 0; i < numArray.length; i++) {
        sum += numArray[i];
    }

    /*Code-Block 3, Expert-Explanation, short and gap-fill questions*/
    average = sum / numArray.length;
    System.out.format("The average is: %.2f", average);
}}
```

3 System Design

3.1 Machine Noun QG

First, Machine Noun QG segments the expert's explanation for each code block into individual sentences using a library called pySBD, a pipeline extension in spaCy v2.0. Then, we extract noun chunks for each sentence, also using spaCy. When a sentence has multiple noun chunks, the first step is to discard any noun chunk with more than four words; Chau and colleagues define "single words or short phrases of two to four words" as domain concepts [4, 3] (i.e., ideally what we would like to target with our questions). Then, we select the longest noun chunk from the remaining noun chunks because longer inputs are beneficial for the question generator. If two noun chunks have the same length, we select the noun chunk that has appeared first in the sentence, assuming that an important keyphrase comes first.

Next, We pass a pair of <sentence, selected noun chunk for the sentence> to a pre-trained sequence-to-sequence model ProphetNet [6] fine-tuned for question generation tasks using the SQUAD [7] dataset. The model outputs the short question. The gap-fill question is created by masking the sentence's noun chunk.

3.2 Machine Verb QG

Machine Verb QG works the same way as Machine Noun QG except it targets verb phrase in the input sentences. We extract verb phrases in a sentence by matching the pattern = ['POS': 'VERB', 'OP': '?', 'POS': 'ADV', 'OP': '*', 'POS': 'AUX', 'OP': '*', 'POS': 'VERB', 'OP': '+'], using Matcher in the spacy library.

4 Evaluation

The two independent annotators (Ph.D. students in Computer Science) annotated a total of 450 questions, each 150 (75 short +75 gap-fill) questions generated by Machine Noun QG, Machine Verb QG, and experts (question in our dataset), using the evaluation criteria as described below. The inter-annotator agreement, measured by Cohen's Kappa, is 0.30, 0.39, 0.71, 0.93, 0.37, 0.37, and 0.91 for grammaticality, semantic correctness, domain relevancy, answerability, helpfulness, recognizability and gap-fill questions, respectively.

We evaluated short questions using the following criteria.

1. **Grammaticality**: Is the question grammatically correct?
2. **Semantic Correctness**: Is the question semantically correct?
3. **Domain Relevancy**: Is the question relevant to the target domain, i.e., does it target a programming concept?
4. **Answerability**: Does the question have a clear answer in the input text?
5. **Helpfulness**: Is the question likely to help the student think about the target concept and produce an answer close to the expert-provided explanation?

6. **Recognizability**: How likely is it that a human generated the question?

The scale for the first two, second two, and last two are 1 (Very Poor) to 5 (Very Good), Yes/No, and 1 (Not Likely) to 5 (Very Likely), respectively.

Each gap-fill question is labeled into one of the following categories.

1. **Good**: asks about key concepts and would be reasonably difficult to answer.
2. **OK**: asks about a) key concept but might be difficult to answer or b) likely key concept (weak concept).
3. **Bad**: asks about 1) an unimportant aspect or 2) has an answer that can be figured out from the context of the sentence.
4. **Acceptable**: OK or Good questions are automatically labeled as acceptable.

5 Results

The overview of quality of short and gap-fill questions is shown in Table 1 and Table 2, respectively. To check whether the difference is significant, we use independent-samples t-tests for the mean score and the Chi-square test of independence for the proportion. We present below a detailed analysis and interpretation of these results in accordance with our research questions.

5.1 Short Questions

Table 1. Performance of Machine Noun QG, Machine Verb QG, and Human on Short-questions. SD= Standard Deviation.

	Machine Noun QG	Machine Verb QG	Human
mean grammaticality	4.51(SD=0.62	4.64(SD=0.48)	4.67(SD=0.48)
mean semantic correctness	4.76(SD=0.46)	4.49(SD=0.80	4.84(SD=0.57)
mean helpfulness	4.27(SD=0.96	3.44(SD=0.96)	4.31(SD=0.77)
mean recognizability	3.49(SD=1.33)	2.76(SD=1.17)	4.49(SD=0.91)
answerability(Yes)%	89.3	54.7	97.3
domain Relevancy (Yes)%	92	89.3	93.3

Both Machine Noun QG and Machine Verb QG generated linguistically well-formed, i.e., grammatically and semantically very good questions with mean scores for grammaticality of 4.51 and 4.64 and semantic correctness of 4.76 and 4.49, respectively.

Likewise, it is possible to automatically generate short questions which are pedagogically sound as measured by domain relevancy, answerability, and helpfulness criterion. The systems generated questions relevant to the domain in program comprehension in an impressive proportion: 92% by the Machine Noun QG and 89.3% by the Machine Verb QG. While the Machine Noun QG produced almost all, i.e., 93% answerable questions, the Machine Verb QG generated slightly more than half, i.e., 54.7% questions that are answerable. The

average helpfulness score of Machine Noun QG questions is 4.27 and, therefore, is likely to help students articulate the expected answer. On the other hand, the Machine Verb QG's average helpfulness score is only 3.44, indicating it may or may not help students scaffold explanation for the code.

Also, it is possible for the system to automatically generate short questions that are indistinguishable from human-generated questions, measured by recognizability. The mean recognizability score for Machine Noun QG is 3.49, indicating that human annotators think humans likely generate these. On the other hand, the mean recognizability score for the Machine Verb QG system is 2.76, which signifies that it at least challenges or makes annotators hard to say who generated the questions, i.e., they think the question has equal chances of being created by human or machine.

Comparison: Compared to human, Machine Noun QG performed comparably; we did not find significant difference in mean or proportion in any criteria. However, Machine Verb QG significantly under-performed to human in helpfulness [t(141.27) = -6.09, p=0.00] and answerablity [χ^2(1, n=150) = 35.12, p = 0.00.] criteria, but, no significant difference in rest of criteria. Between machines, Machine Noun QG significantly outperformed machine verb in semantic correctness [t(118.62)=2.51, p=0.01] and helpfulness [t(148) = 5.26, p=0.00], and they pefromed similalry in rest of criterion.

5.2 Gap-Fill Questions

Table 2. Performance of Machine Noun QG, Machine Verb QG, and Human on Gap-Fill Questions.

	Bad %	Okay %	Good %	Acceptable%
Machine Noun QG	16	73.3	10.7	84
Machine Verb QG	20	38.7	41.3	80
Human	2.7	53.3	44	97.3

These systems can produce a majority of acceptable gap-fill-questions, i.e., 84% by Machine Noun QG and 80% by Machine Verb QG.

Comparison: Compared to human, both Machine Noun QG [χ^2(1, n=150)=6.38, p =0.012] and Machine Verb QG [χ^2(1, n=150)=9.55, p =0.002] significantly under-performed in gap-fill QG task. There is no significant difference in the proportions of acceptable gap-fill questions generated between Machine Noun QG and Machine Verb QG, χ^2(1, n=150)=0.19, p=0.67.

6 Conclusion

In this work, we developed Machine Noun QG and Machine Verb QG systems to automatically generate short and gap-fill questions that ITS can use to scaffold students by presenting them as a hint. Our evaluation shows that these

systems can generate short questions which are linguistically well-formed, pedagogically sound, and likely indistinguishable from human-generated questions. We also found that most gap-fill questions generated by machines are of acceptable quality to be used by ITS. Compared to human experts, Machine Noun QG performed comparable for short questions but under-performed for gap-fill questions in almost all criteria. Between the systems, Machine Noun QG performed better.

In our future work, we plan to automate the generation of code explanations using code examples and the surrounding text in programming textbooks and then use the explanations to generate the questions automatically, thus making the process fully automated.

References

1. Aleven, V., Mclaren, B.M., Sewall, J., Koedinger, K.R.: A new paradigm for intelligent tutoring systems: Example-tracing tutors. International Journal of Artificial Intelligence in Education **19**(2), 105–154 (2009)
2. Alshaikh, Z., Tamang, L.J., Rus, V.: Experiments with auto-generated socratic dialogue for source code understanding. In: CSEDU (2). pp. 35–44 (2021)
3. Banjade, R., Oli, P., Tamang, L.J., Chapagain, J., Rus, V.: Domain model discovery from textbooks for computer programming intelligent tutors. In: The International FLAIRS Conference Proceedings. vol. 34 (2021)
4. Chau, H., Labutov, I., Thaker, K., He, D., Brusilovsky, P.: Automatic concept extraction for domain and student modeling in adaptive textbooks. International Journal of Artificial Intelligence in Education **31**(4), 820–846 (2021)
5. Hsiao, I.H., Brusilovsky, P., Sosnovsky, S.: Web-based parameterized questions for object-oriented programming. In: E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education. pp. 3728–3735. Association for the Advancement of Computing in Education (AACE) (2008)
6. Qi, W., Yan, Y., Gong, Y., Liu, D., Duan, N., Chen, J., Zhang, R., Zhou, M.: Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. arXiv preprint arXiv:2001.04063 (2020)
7. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250 (2016)
8. Tamang, L.J., Alshaikh, Z., Khayi, N.A., Oli, P., Rus, V.: A comparative study of free self-explanations and socratic tutoring explanations for source code comprehension. In: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. pp. 219–225 (2021)
9. Tamang, L.J., Alshaikh, Z., Khayi, N.A., Rus, V.: The effects of open self-explanation prompting during source code comprehension. In: The Thirty-Third International Flairs Conference (2020)
10. Thomas, A., Stopera, T., Frank-Bolton, P., Simha, R.: Stochastic tree-based generation of program-tracing practice questions. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education. pp. 91–97 (2019)
11. Zavala, L., Mendoza, B.: On the use of semantic-based aig to automatically generate programming exercises. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. pp. 14–19 (2018)