CS1QA: A Dataset for Assisting Code-based Question Answering in an Introductory Programming Course

Changyoon Lee, Yeon Seonwoo, Alice Oh
School of Computing, KAIST
changyoon.lee@kaist.ac.kr, yeon.seonwoo@kaist.ac.kr
alice.oh@kaist.edu

Abstract

We introduce CS1QA, a dataset for code-based question answering in the programming education domain. CS1QA consists of 9,237 question-answer pairs gathered from chat logs in an introductory programming class using Python, and 17,698 unannotated chat data with code[1]. Each question is accompanied with the student's code, and the portion of the code relevant to answering the question. We carefully design the annotation process to construct CS1QA, and analyze the collected dataset in detail. The tasks for CS1QA are to predict the question type, the relevant code snippet given the question and the code and retrieving an answer from the annotated corpus. Results for the experiments on several baseline models are reported and thoroughly analyzed. The tasks for CS1QA challenge models to understand both the code and natural language. This unique dataset can be used as a benchmark for source code comprehension and question answering in the educational setting.

1 Introduction

Question answering (QA) studies systems that understand questions and the relevant context to provide answers. Question forms include single document QA (Rajpurkar et al., 2016), multi-hop QA (Yang et al., 2018), conversational QA (Reddy et al., 2019), and open domain QA (Kwiatkowski et al., 2019). Questions about specific domains are asked in NewsQA (Trischler et al., 2016) and TechQA (Castelli et al., 2020), and images are provided with the question in visual QA (Antol et al., 2015). Another interesting field of QA asks questions about source code (Liu and Wan, 2021).

A useful application of QA is in the educational domain. Asking questions and getting the answer is an essential and efficient means of learning. In this paper, we focus on QA for programming education,

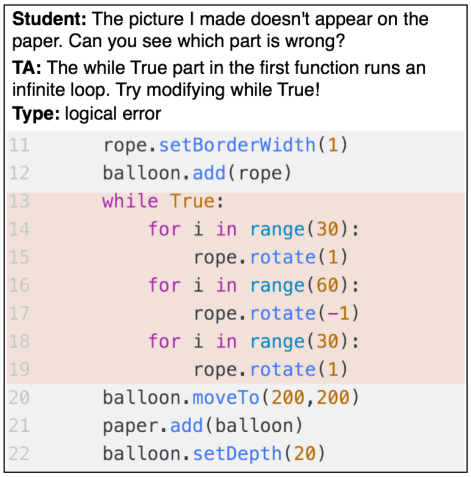

Figure 1: An example of our data tuple. Each data tuple consists of {question, answer, question type, code, relevant code lines}. We annotate the type of each question and the code lines (orange) relevant to the question.

where both the input modes and the domain pose interesting challenges. Answering these questions requires reading and understanding both source code and natural language questions. In addition, students' questions are often complex, demanding thorough understanding of the context such as the intention and the educational goal to answer them.

Recently, models that understand programming languages (PL) have been studied, and show promising results in diverse code comprehension tasks (Alon et al., 2018; Feng et al., 2020; Guo et al., 2021). However, these models have limitations to support question answering. They are not trained on datasets containing questions about the code and are not designed for QA tasks. Also, many assume fully functional code as input, while students' code contains diverse syntax and logical errors and is often incomplete.

To address this issue, we introduce CS1QA, a new dataset with tasks for code-based question answering in programming education. Questions and answers about programming are collected from the

[1]The code and the data used in this paper can be found at https://github.com/cyoon47/CS1QA.

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2026 - 2040

naturally occurring chat messages between students and TAs. The question type and the code snippet relevant to answering the question are also collected. The final CS1QA dataset consists of question, question type, answer, and code annotated with relevant lines. The data is collected mostly in Korean and then machine-translated into English and quality-checked for easy application on models pretrained in English. Figure 1 shows an example of our data. We also include two-semesters' worth of TA-student chat log data consisting of 17,698 chat sessions and the corresponding code.

We design three tasks for the CS1QA dataset. Type classification task asks the model to predict the question type. Code line selection task asks the model to select lines of code that are relevant to answering the given question. Answer retrieval task finds a similar question already answered, and uses its answer as the answer to the given question. The outputs for these tasks can help the students debug their code and the TAs spend less time and effort when answering the students' questions.

Finally, we implement and test baseline models, RoBERTa (Liu et al., 2019), CodeBERT (Feng et al., 2020) and XLM-RoBERTa (Conneau et al., 2020), on the type classification and code line selection tasks. The finetuned models achieve accuracies up to 76.65% for the type classification task. The relatively low F1 scores of 57.57% for the line selection task suggest that the task is challenging for current language models. We use DPR (Karpukhin et al., 2020) to retrieve the most similar question and its answer. We compare the retrieved answer with the gold label answer, and achieve a BLEU-1 score of 13.07, which shows incompetent performance of answer retrieval on CS1QA dataset. We show with a qualitative evaluation the model behavior with different inputs for the first two tasks. Our contributions are as follows:

- We present CS1QA, a dataset containing 9,237 question-answer-code triples from a programming course, annotated with question types and relevant code lines. The dataset's contribution includes student-TA chat logs in a live classroom.

- We introduce three tasks, question type classification, code line selection and answer retrieval, that require models to comprehend the text and provide useful output for TAs and students when answering questions.

- We present the results of baseline models on the tasks. Models find the tasks in CS1QA challenging, and have much room for improvement in performance.

2 Related Work

Code-based Datasets Recently, research dealing with large amounts of source code data has gained attention. Often, the source code data is collected ad hoc for the purpose of the research (Allamanis et al., 2018; Brockschmidt et al., 2018; Clement et al., 2020). Several datasets have been released to aid research in source code comprehension, and avoid repeated crawling and processing of source code data. These datasets serve as benchmarks for different tasks that test the ability to understand code. Such datasets include: ETH Py150 corpus (Raychev et al., 2016), CodeNN (Iyer et al., 2016), CodeSearchNet (Husain et al., 2020) and CodeQA (Liu and Wan, 2021). We compare these datasets with CS1QA in Table 1.

In an educational setting, students' code presents different chracteristics from code in these datasets: 1) students' code is often incomplete, 2) there are many errors in the code, 3) students' code is generally longer than code used in existing datasets, and 4) questions and answers from students and TAs provide important additional information. In CS1QA, we present a dataset more suited for the programming education context.

Source Code Comprehension In the domain of machine learning and software engineering, understanding and representing source code using neural networks has become an important approach. Different approaches make use of different characteristics present in programming languages. One such characteristic is the rich syntactic information found in the source code's abstract syntax tree (AST). Code2seq (Alon et al., 2018) passes paths in the AST through an encoder-decoder network to represent code. The graph structure of AST has been exploited in other research for source code representation on downstream tasks such as variable misuse detection, code generation, natural language code search and program repair (Allamanis et al., 2018; Brockschmidt et al., 2018; Guo et al., 2021; Yasunaga and Liang, 2020). Source code text itself is used in models such as CodeBERT (Feng et al., 2020), CuBERT (Kanade et al., 2020) and DeepFix (Gupta et al., 2017) for use in tasks such

Dataset	Programming Language	Data Format	Dataset Size	Data Source
ETH Py150	Python	Parsed AST	7.4M files	GitHub
CodeNN	C#, SQL	Title, question, answer	∼187,000 pairs	StackOverflow
CodeSearchNet	Go, Java, JavaScript, PHP, Python, Ruby	Comment, code	∼2M pairs	GitHub
CodeQA	Java, Python	Question, answer, code	∼190,000 pairs	GitHub
CS1QA	Python	Chat log, question, answer, type, code	9,237 pairs	Real-world classroom

Table 1: Comparison between different code-based datasets and CS1QA.

as natural language code search, finding function-docstring mismatch and program repair.

The tasks that these methods are trained on target expert software engineers and programmers who can gain significant benefit with support by the model. On the other hand, students learning programming have different objectives and require fitting support by the models. Rather than getting an answer quickly, students seek to Students ask lots of questions while learning, and thus question answering for code is needed. CS1QA focuses on code-based question answering and can be used as training data and a benchmark for neural models in an education setting. The CS1QA data can also be used for other tasks than QA, such as program repair and code search.

3 CS1QA Dataset

3.1 Data Source

The data for CS1QA is collected from an introductory programming course conducted online. Students complete lab sessions consisting of several programming tasks and students and TAs ask questions to each other using a synchronous chat feature. We make use of the chat logs as the source for the natural question and the corresponding answer. These chat logs are either in Korean or in English. The student's code history is also stored for each programming task for every keystroke the student makes. This allows us to extract the code status at the exact time the question is asked, which provides valuable context for the question. We take this code as the context for the given question. The thorough code history and the student-TA chat logs are a unique and important contribution of CS1QA. CS1QA also contributes with data from multiple students working on the same set of problems.

3.2 Question Type Categorization

Answering different types of questions requires understanding the different intentions and information - answering questions about errors requires identifying the erroneous code and answering questions about algorithms requires understanding the overall program flow. As the different question types affect the answering approach and location of code to look at, knowing them in advance can be beneficial in the QA and code selection tasks.

Allamanis and Sutton (2013) have categorized questions asking for help in coding on Stack Overflow into five types. We adapt these types to students' questions. In addition, we define the "Task" type that asks about the requirements of the task. TAs' question types are derived from the official instructions by the course instructors given in the beginning of the semester. TAs were instructed to ask questions that gauge students' understanding of their implementation, for example by asking the meaning of the code and reasoning behind the implementation. TAs' probing questions are categorized into five types: *Comparison*, *Reasoning*, *Explanation*, *Meaning*, and *Guiding*. Examples for the question types can be found in Table 2. We present intentions of the question types in Table 3.

3.3 Collecting Question-Answer Pairs with Question Types

We collected a total of 5,565 chat logs over the course of one semester from 474 students and 47 TAs. After removing the logs where the TA did not participate in the chat, 4,883 chat logs remained.

We employed crowdworkers with self-reported skill in Python of three or higher on a 5-point Likert Scale to collect the questions. Each worker first selected messages in the chat log corresponding to the question and the answer, then selected the question type. Workers were provided with descriptions of the question types with examples before working on the task. Workers were asked to divide the message into individual questions when there were multiple questions or answers in the message. They were instructed to only choose programming related questions, for which the answer is obvious in the chat from the question alone. This ensures that the questions and answers are independent from the

Question Type	Allamanis' Type	Question	Answer
Code Understanding	How/why something works	Why is the print cards function at the bottom of the check function? Can I not have it?	This is because if two cards match through a check, you have to show them two cards.
Logical Error	Do not work	Now, the file is created, but when I go inside and look at the value, it seems to be a little different from the one requested in the problem.	You seem to have forgotten the line break in the middle I think you can add \n
Error	Do not work	I don't know where the task 2 error came from....	When creating image There is a negative number in the image size Please do something like absolute value
Function/Syntax Usage	Way of using	And I forgot how to make a blank image	Blank images can be created with new_img= create_picture(width,height)!
Algorithm	How to implement	So, what if there is any other way to count the number including the number not included in the randomly created list?	Parameter: a list which is returned from drawing_integers Count integers function is supposed to take that as input
Task	-	Isn't it a task in which the number of iteration steps changes according to the input value?	Yes, but the value of x must also change according to the input value!
Comparison	-	How was the method of reading and writing different in task1?	There was a difference between read mode and write mode, open(file name, r) and open (file name, w)
Reasoning	-	I've read the task1 code. What is the intention of using continue on line 55?	This is to go back to the beginning without executing the next print function.
Explanation	-	How do you create new_img when 'horizontal' is input as Direction in Task2?	I did it like I did with vertical, but since the y value is changing, when I change the y value and run the loop, I did x first among x and y.
Meaning	-	Can you explain the role of the global keyword in Task 1?	If you use a variable specified only within a function, it cannot be used in other functions, so I used it for global variable processing!
Guiding	-	Is there a simpler way to change average_integers using a function already defined in the python list??	In average_integers, it would be more convenient to use the len function when counting the total number of elements.

Table 2: Examples of translated and untranslated question and answer texts for each question type in CS1QA. First column shows our type classification, and second column shows the classification by Allamanis and Sutton (2013). The first six rows in the top part are student question types, the last five rows in the bottom part are TA's probing question types.

chat history. Every chat log was annotated by two workers to ensure the quality of annotation. A total of 20,403 question-answer pairs were collected.

The question and answer texts are machine-translated using Google's Neural Machine Translation model (Wu et al., 2016) from Korean to English to form the dataset in two languages. The translation allows for easy integration of CS1QA data to models pretrained in English, which make up a huge portion of NLP models.

3.4 Selecting Code Lines

Providing relevant code snippets allows the answerer to identify the problem more quickly and easily. We annotate the lines of code that provide information necessary to answer the question for use in the code line selection task.

We collected code for all questions asked by students. For the TA probing questions, we collected code for all *Reasoning* and *Meaning* types, and 472 randomly selected *Explanation* questions, for a total of 4677 questions. This keeps a balance in the number of questions for each type. *Comparison* questions were left out as they require comparison of code across different tasks, making the annotation and the tasks too complicated. We exclude *Guiding* questions as answering them requires more than just understanding code; the answer is often new algorithms not based on the current code.

We employed crowdworkers who have worked as TAs for the programming course to select the code that the questions refer to. We provided the workers with the collected questions, answers, and the student's code for each task at the time the question was asked. The workers selected the code file for the question and the relevant code lines to answer the question. When reading the code was not necessary to answer the question, the workers were asked to choose *Not Applicable (NA)* for the code selection. For every question, two workers made code annotations.

A total of 9,359 code selections were made by the workers. Some of the selections with empty code or incorrectly extracted code were removed from the dataset. The remaining 9,237 questions annotated with type, lab and task numbers, code, code lines and answer make up the final CS1QA dataset. Every code selection made by the workers is used as gold labels even if the two workers choose different lines. Thus, every question can have up to two correct code selections in different parts of code. An example of the data is found in Appendix A.

	Q Type	Intention	# Q	# Code	NA (%)	Span (%)
S	Code Understanding	Understanding the functionality of the code	105	209	33.9	11.0
	Logical Error	Investigating the cause of the unexpected outputs of the code	541	1060	16.7	21.6
	Error	Resolving syntax errors and exceptions	488	959	10.1	13.0
	Function/ Syntax Usage	Learning correct usage of a function or syntax	411	811	55.4	11.8
	Algorithm	Learning the underlying algorithm for the task	603	1194	50.0	19.3
	Task	Confirming the goal and requirement of the task	677	1322	82.3	15.3
T	Reasoning	Understanding the reasoning behind student's implementation	402	799	5.6	21.8
	Explanation	Checking the validity of student's explanation of their code	472	940	2.6	42.8
	Meaning	Checking the meaning of a function or a variable in the code	978	1943	2.1	24.2

Table 3: Types of questions asked by students (S) and TAs (T) in a programming class. Questions are categorized by different intention and information required to answer them. The number of questions and code snippets collected from the annotators, percentage of *Not applicable* code selections and selected code lines are reported.

3.5 Quality Control and Validation

As the workers worked independently, there were some differences in the annotated data even when they correspond to the same question. There were some questions that were selected by only one worker as well. These questions are further reviewed to ensure the quality of the collected dataset.

Out of 20,403 collected questions, 3,556 questions were selected by only one worker, and 4,787 pairs of questions had some differences between the workers' selections. The remaining questions had perfect agreement between the workers. The authors reviewed questions selected by only one worker, and those without perfect agreement. Unnecessary words present in only one text were removed and crucial words missing in the question were added to the text while preserving the meaning to make the two texts equal. The conflicts in question types were resolved with the authors' additional vote that made a clear majority in the type selection.

We calculate the inter-rater reliability score with Cohen's Kappa (Cohen, 1960) for the question type selection. The Kappa value is calculated between every pair of workers who selected the same question-answer pairs. The mean of the Kappa values is 0.657, which suggests substantial agreement for type classification between the annotators.

Out of 9,237 questions with code line selection, 2,197 pairs had perfect agreement (100% overlap), while 1,225 pairs had 0% overlap. We compute the mean line F1 as the measure for agreement of spans, considering one annotator's span selection as the ground truth and the other annotator's selection as prediction. The resulting F1 score is 0.6482. The disagreements are largely due to selecting different but relevant code and selecting different amounts of surrounding context in the code.

4 Task Definition

We design three tasks for the CS1QA dataset that identify important information that leads to the answer.

The type classification task is to predict the question's type. We use nine types of questions that we categorized as the candidates for classification, each question belonging to a single type. We use the accuracy and macro F1 score as the measure of performance. The code line selection task is to select lines of code that give relevant information to answer the question. The code is a strong supporting context to answering the given question, and this task tests the model's ability to retrieve this critical information.

For the code line selection task, we use the Exact Match (EM) and line F1 score as the measure of performance, same as the metrics used for supporting fact selection task in HotpotQA (Yang et al., 2018). The EM score measures the proportion of selections that exactly match the ground truth. The line F1 score measures the average overlap between the selected lines and the ground truth selections. The score is computed by treating the selections as bags of lines and calculating their F1 with the annotated lines. These two tasks take as inputs the lab

and task numbers of the question, the questioner (student or TA), question and the code texts.

The answer retrieval task retrieves a similar question given an unseen question, and uses the retrieved question's answer as the answer to the unseen question. BLEU score is calculated between the retrieved answer and the gold label answer.

Answer generation task given the question and the code context is possible with CS1QA dataset. However, meaningfully generating the answer demands a model that understands long and erroneous code, and the natural language question. This poses a significant challenge, and we leave the generation task as future work.

5 Dataset Analysis

644 out of 9,237 questions are originally asked in English, while the rest are asked in Korean. The CS1QA dataset is split into train, development and test sets in the ratio of 0.6, 0.2 and 0.2 respectively, keeping the ratio of question types in each set the same to ensure equal distribution in all three sets.

5.1 Text Lengths

Table 4 shows the statistics of question and answer token lengths, for data translated to English (EN) and the original (ORIG) data, and the number of lines of code.

	Data	Min	Max	Mean	Median
Question	EN	1	119	15.7	13
	ORIG	1	79	10.9	9
Answer	EN	1	272	27.2	22
	ORIG	1	166	17.6	14
Code	-	1	655	76.0	52

Table 4: Statistics of question, answer lengths in tokens and code length in number of lines in CS1QA.

The lengths of questions and answers lie mostly between 10 to 30 tokens. The distributions show long tails for both questions and answers, but answers are more evenly distributed. The distribution of token lengths for questions and answers can be found in Appendix B.

The number of lines of code shows a peak between 12.5 and 50, as shown in Figure 2. Code snippets have a wider distribution in length. This can be the result of varying difficulties of tasks, with more difficult tasks requiring longer code snippets to solve. The number of lines of code in CS1QA is larger than those in other code-based datasets, which can present interesting challenges.

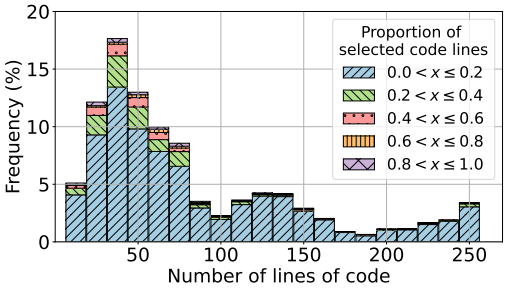

Figure 2: The total number of lines in code and the proportion of selected code lines in code (color coded). The last bin contains all code longer than 250 lines.

5.2 Question Type Distribution

We present the distribution of question types in Table 3 in number of questions collected and number of code snippets collected.

The CS1QA dataset contains similar number of questions for each student type of questions, except for *Code Understanding* type, which contains significantly fewer questions. One plausible reason for this is that most of the tasks require writing the program from scratch, thus students ask fewer questions about the skeleton code.

There are more *Meaning* questions than other types of TAs' probing questions. This can be because TAs often ask the students about the meanings of functions and variables to make sure that the students understand the code they wrote for each task.

5.3 Code Line Distribution

The average number of selected code lines is 13.0. A majority of the questions can be answered by looking at fewer than 20 lines of code. The number of selected code lines can be a gauge of the difficulty of answering the questions; a longer selection means that one has to read and understand a larger amount of code. The detailed distribution of code lines and code lengths can be found in Appendix B. Figure 2 shows the percentage of selected code lines. The graph shows that majority of the selected code lines are less than 20% of the total number of lines of code.

The proportion of *Not applicable* code selections differ by question types, as shown in Table 3. As TAs ask questions about the implementation details, answering most of them requires looking at the code. On the other hand, students often ask about

the approach to the problems and implementation. These questions have less basis on the code and often refer to shorter spans where an error occurs or a function is used. Thus finding the relevant code takes more effort, although answering them requires looking at less code on average.

5.4 Machine Translation Quality

We have employed 16 workers who are fluent in both Korean and English to check the quality of the machine translation of sampled questions and answers. Each worker checked the quality of 8 question-answer pairs per question type. Each pair was checked by at least two workers. Workers compared the original and the translated texts, and gave scores to four statements on a 5-point Likert scale, with 1 being disagree/bad and 5 being agree/good. The statements were: 1) I can understand the translation, 2) The translation has similar meaning to the original text, 3) The translation contains grammatical and lexical errors, and 4) Overall translation quality. The mean scores between workers for each statement were 4.37, 4.11, 2.06 and 3.92 respectively. The results suggest that the translation was overall in good quality, with high understandability and similar meaning to the original text. The translation contains grammatical or lexical errors, but not to a significant extent.

6 Experimental Setup

We select three baseline models, CodeBERT, RoBERTa and XLM-RoBERTa, and test their performance on the type classification and code line selection tasks. CodeBERT model is selected to test the effectiveness of pretraining on NL-PL paired data. Other models based on syntactic structures of code cannot take students' erroneous code as input. RoBERTa and XLM-RoBERTa models are selected to test the performance of NL-based models, for translated and untranslated data respectively. Questions translated to English are provided to the two models pretrained in English, CodeBERT and RoBERTa. XLM-RoBERTa model receives the untranslated questions as input to compare the performance when using the untranslated data. We used the default hyperparameters used in CodeBERT for training. The tokenizers encode newline token to maintain the code's structure in the tokenized text. For the code line task, we also test the performance of the naive baseline, which selects the middle 60 lines of code, which showed the best performance among different numbers of lines, as the output.

Since the token lengths for code in CS1QA are greater than the limit of the transformer-based models, we preprocess the input to fit within the token length limit. We split the code into smaller segments so that the combined length of the split segment and the question is within the limit. For type classification, the type with the most number of votes is selected as the final selection. For code line selection, the model chooses a start and end token position from each segment. The lines between the start and end tokens are given as the output for the segment, and the union of segment outputs is given as the final selection for the question. N/A is given as the output when 1) the end position is before the start position, 2) either the start or the end position is 0 ([CLS] token), or 3) either the start or the end position is out of range.

For the answer retrieval task, we train the DPR by taking the questions as the passages. We use the question with the highest BM25 score in the corpus set as the gold label for the questions in the training set. For testing, the most similar question in the corpus is retrieved using the trained DPR with the new question as the query. The retrieved answer is used as the answer to the new question verbatim.

7 Results

We report the mean score from three runs with different seeds for all experiments. The test score is reported on the best-performing epoch out of 10 on the development set.

7.1 Type Classification

The results of our baseline models on type classification are shown in Table 5. The models learn to predict the question types with relatively high accuracy, but there is still a room for improvement.

Model	Dev		Test		Q only	
	Acc	F1	Acc	F1	Acc	F1
RoBERTa	**77.57**	**72.31**	**76.65**	**71.10**	75.74	**69.40**
CodeBERT	76.20	69.09	75.65	70.13	74.75	67.07
XLM-R	72.60	67.88	72.62	66.19	**76.18**	68.68

Table 5: Type classification task scores for the three baseline models. Q only column shows the test scores with only the question text as the input.

The class-wise classification F1 scores in Table 6 shows a significant drop for 'understanding' type when code is not provided. The low number of

		Understanding	Logical	Error	Usage	Algorithm	Task	Reasoning	Explanation	Meaning
RoBERTa	w/ code	29.26	70.53	77.14	53.23	60.22	66.95	96.41	91.59	95.26
	w/o code	21.99	70.76	76.39	50.87	67.96	68.64	95.32	88.35	94.38
CodeBERT	w/ code	28.80	68.35	74.46	54.29	61.77	66.26	95.94	87.87	93.80
	w/o code	13.63	67.91	74.77	44.07	59.93	65.49	95.96	87.91	93.98
XLM-R	w/ code	15.12	68.13	72.26	46.87	59.22	61.88	93.83	86.32	92.12
	w/o code	9.70	71.40	75.20	53.04	61.42	67.05	97.44	88.77	94.12

Table 6: Class-wise F1 scores on test set for type classification for baseline models

		Understanding	Logical	Error	Usage	Algorithm	Task	Reasoning	Explanation	Meaning
RoBERTa	w/ code	30.89	72.64	77.09	53.18	59.95	68.06	96.96	90.62	95.00
	w/o code	33.16	72.67	77.45	53.37	59.46	66.40	96.15	90.05	94.44
CodeBERT	w/ code	28.93	68.82	76.87	54.53	60.02	64.35	95.94	87.42	94.81
	w/o code	30.33	65.67	72.85	54.65	60.97	67.60	96.19	88.48	94.96
XLM-R	w/ code	25.02	73.44	76.80	52.77	62.13	67.21	95.90	87.92	94.21
	w/o code	28.77	69.68	77.39	54.83	59.23	67.29	96.70	88.17	93.91

Table 7: Class-wise F1 scores on test set for type classification for baseline models trained with augmented data.

questions for the understanding type might be the reason, thus we augment the dataset with generated understanding type questions. The common question templates for understanding type questions are extracted, and keywords in the question are randomly replaced with keywords in a randomly chosen code in the dataset. The generated question and the chosen code are given as the input to the models. The question templates are provided in Appendix C. The class-wise classification F1 scores are reported in Table 7. The difference in scores depending on the presence of code is reduced, and overall performance increases. The results suggest that presence of code does not significantly affect the type classification performance. This is expected, as the question type annotation was conducted without providing the code.

7.2 Code Line Selection

The results of our baseline models on line selection are shown in Table 8. We also conduct another set of experiments with questions with N/A line selection removed (Valid Line column). The drop in scores on the code with valid line selections shows that large portion of the scores come from the model correctly identifying N/A selections.

The naive baseline performance is much worse than the models' performance, which suggests that line selection task is not trivially solved. The relatively low scores on the tasks for CS1QA suggest that they are challenging for models built for natural language understanding. CodeBERT's performance is not superior for the span selection task even though the model was pretrained on code and natural language together. This suggests that CodeBERT's pretraining objective is not appropriate for the CS1QA tasks.

Model	Dev		Test		Valid Line	
	EM	F1	EM	F1	EM	F1
Naive	1.08	23.97	0.65	21.84	0.90	30.42
RoBERTa	**46.62**	**62.61**	**41.80**	**57.57**	**22.02**	43.50
CodeBERT	42.00	57.74	38.95	54.06	16.42	37.12
XLM-R	42.57	58.63	39.14	55.40	21.85	**43.90**

Table 8: The naive and three baseline models' scores on line selection task.

7.3 Answer Retrieval

The mean BLEU-1 score that compare the answers for the questions in the test set is 13.07. This shows that a simple retrieval based answering system is not sufficient for answering students' questions. The code provides important context to generate accurate answers, and the answer likely differs even for the same question, depending on the code.

The mean BLEU score for TA's probing questions is 18.48, while that for student-asked questions is 8.91. This suggests that the TAs tend to ask similar questions that have similar answers, while students' questions vary more with largely different answers.

7.4 Qualitative Analysis

In order to better understand the baseline models' behavior, we analyze the output type classifications and line selections for 180 questions, 20 per question type.

For type classification, most of the 'why' questions from students are classified as 'logical error' or 'error' types. These questions are often phrased as "I don't know why..." or "why something doesn't work". This leads to relatively high scores for the two error types. 84% of the 'why' questions were classified into the two types. 15 questions were correctly classified.

Keyword matching for line selection task accounts for approximately 54% of line selections. When a function name or variable name is mentioned in the question, the selected code lines often include the mentioned name. However, this tactic sometimes fools the model into selecting more lines than necessary. This was more frequently observed for *Meaning* and *Function/Syntax Usage* tasks, where 94% and 75% of the line selections included the keyword.

8 Conclusion

In this paper, we present CS1QA, a dataset for code-based question answering in introductory programming course. CS1QA's crowdsourced data from a programming course provide rich information that code understanding models need to consider to correctly answer the given questions. We introduce three tasks for CS1QA, whose output can help students debug and reduce workloads for the teaching staff. Results from the baseline models indicate that tasks for CS1QA are challenging for current language understanding models. CS1QA promotes further research to better represent and understand source code for code-based question answering.

As CS1QA data deliver the full context of the questions, the answer texts in CS1QA can be used as training and testing data for an answer generation task in the future. Although the generation task is difficult and demands new code representation and processing methods, models that show good performance on it will allow a new level of automation in code-based QA. We hope that CS1QA will bring research interest in the domain of code understanding for question answering.

9 Ethical Consideration

All students and TAs, whose chat logs and code are used to build the dataset, have given permission to use these data for research purposes prior to this research. No disadvantage was given to any student or TAs for not providing their data for this research. The IRB at our university approved the annotation experiments conducted in this research.

The annotators were compensated appropriately for their participation in the experiments. Compensation was determined to meet the minimum wage requirements. For the experiment collecting question-answer pairs with question types, the workers were paid $9 for the first 50 chat logs marked and $13.50 for every 50 chat logs marked afterwards. It took less than an hour to complete annotations for 50 chat logs on average. For the experiment collecting the code lines, the workers were paid $0.45 for every code annotation made. Workers were able to complete approximately 30 selections in an hour on average. For the experiment testing the effectiveness of providing relevant code lines on answering the questions, the participants were paid $13.50 to answer 48 questions by the students. It took approximately an hour for each participant to finish answering all 48 questions.

The authors made their best efforts to anonymize the dataset and remove all personal information such as student ID and phone number from the dataset.

References

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to represent programs with graphs. In *International Conference on Learning Representations*.

Miltiadis Allamanis and Charles Sutton. 2013. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. In *International Conference on Learning Representations*.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*.

Marc Brockschmidt, Miltiadis Allamanis, Alexander L Gaunt, and Oleksandr Polozov. 2018. Generative code modeling with graphs. In *International Conference on Learning Representations*.

Vittorio Castelli, Rishav Chakravarti, Saswati Dana, Anthony Ferritto, Radu Florian, Martin Franz, Dinesh Garg, Dinesh Khandelwal, Scott McCarley, Michael McCawley, Mohamed Nasr, Lin Pan, Cezar Pendus, John Pitrelli, Saurabh Pujar, Salim Roukos, Andrzej Sakrajda, Avi Sil, Rosario Uceda-Sosa, Todd Ward, and Rong Zhang. 2020. The TechQA dataset. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online. Association for Computational Linguistics.

Colin Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. Pymt5: Multi-mode translation of natural language and python code with transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1).

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Édouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcode{bert}: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixing common c language errors by deep learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2020. Codesearchnet challenge: Evaluating the state of semantic code search.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*. PMLR.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7.

Chenxiao Liu and Xiaojun Wan. 2021. Codeqa: A question answering dataset for source code comprehension. In *Findings of the Association for Computational Linguistics: EMNLP 2021*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas. Association for Computational Linguistics.

Veselin Raychev, Pavol Bielik, and Martin Vechev. 2016. Probabilistic model for code with decision trees. *ACM SIGPLAN Notices*.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics*, 7.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2016. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean.

2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Michihiro Yasunaga and Percy Liang. 2020. Graph-based, self-supervised program repair from diagnostic feedback. In *International Conference on Machine Learning*, pages 10799–10808. PMLR.

Appendix

A Example data in CS1QA

We present an example of a question in the CS1QA dataset in Figure 3.

```
labNo: 6,
taskNo: 0,
questioner: “student”
question: "I wrote a Fibonacci function and ran it, but the Fibonacci sequence was not input, but only [] appeared. It
          comes out as [], not None.",
code:
“def fibonacci(upper_bound):
        i = 0
        k = 1
        fibo_list = []
        while i > upper_bound:
            fibo_list.append(i)
            fibo_list.append(k)
            i = i + k
            k = i + k
        return fibo_list

    print(fibonacci(1000))”,
startLine: 4,
endLine: 4,
questionType: “logical”
answer: "It looks like the direction of the inequality sign in the while statement is wrong."
```

Figure 3: An example of the data in CS1QA. Note that taskNo, startLine and endLine variables count from 0. The code is prettified for readability.

B Distribution of Question, Answer and Code Lengths

Figures 4 and 5 show the distribution of question lengths for questions translated to English and original questions respectively.

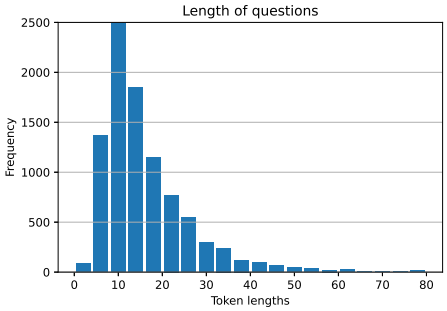

Figure 4: The distribution of question lengths translated to English in number of white space separated tokens. The last bin contains all questions longer than 80 tokens.

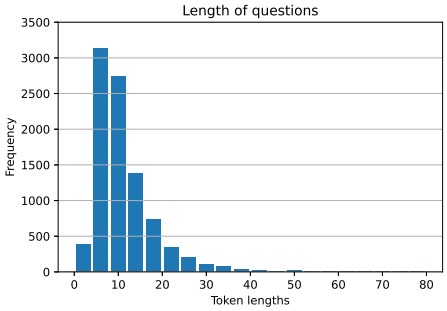

Figure 5: The distribution of original question lengths in number of white space separated tokens. The last bin contains all questions longer than 80 tokens.

Figures 6 and 7 show the distribution of answer lengths for answers translated to English and original answers respectively.

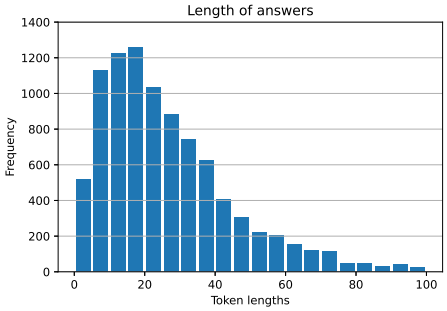

Figure 6: The distribution of answer lengths in number of white space separated tokens. The last bin contains all answers longer than 100 tokens.

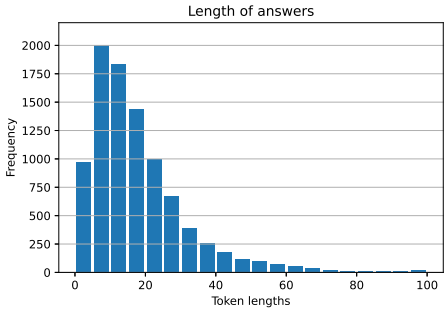

Figure 7: The distribution of original answer lengths in number of white space separated tokens. The last bin contains all answers longer than 100 tokens.

Figure 8 shows the distribution of number of lines in the selected code spans. Figure 9 shows the distribution of proportions of the code lines that is included in the selected code span.

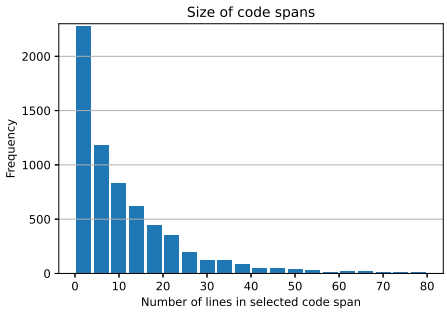

Figure 8: The distribution of number of lines selected in code spans. The last bin contains all selections with more than 80 lines of code.

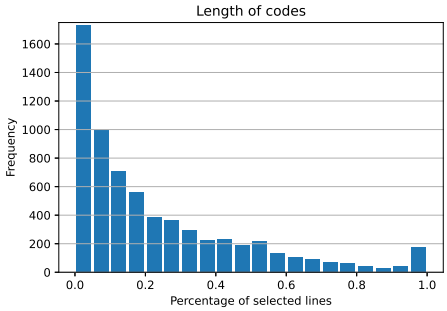

Figure 9: The distribution of the percentage of selected code lines in code.

C Question Templates for Understanding Type Questions

The templates used for augmenting CS1QA dataset with understanding type questions are presented in Table 9. The keywords variable, function, and snippet are extracted from the randomly chosen code in the dataset. Variable is a random token, function is a random function name, and snippet is a random line of code. In the last template, one of the words list, dictionary, variable, function is chosen randomly to complete the template.

Template format
What does [variable, function] mean?
What does [variable, function] refer to?
What's the meaning of [variable, function]
What does [function] do?
Can you explain what [function] does?
Can you describe what [function] is doing?
How do I use [function]?
How to use [function]?
I don't understand [snippet].
What is [function]?
Should I use [function, snippet]?
Why do you do [snippet]?
Is [variable, function] a {list, dictionary, variable, function}?

Table 9: Templates used for the question augmentation for Understanding type questions. The keywords in square brackets are chosen from a randomly chosen code in the dataset. The words in curly brackets are randomly chosen.

D Experiment Details

We ran the experiments for RoBERTa-base, CodeBERT-base and XLM-RoBERTa model on 4 Quadro RTX 8000 GPUs. We ran 10 epochs for fine-tuning the models. All of these models were released with MIT License, and our use is consistent with the license.

For all models, we used the batch size of 32 for training, evaluating and testing.

The average runtime for each epoch for RoBERTa-base and CodeBERT-base models is approximately 1 hour for training, and 1 minute for evaluating and testing. For XLM-RoBERTa-base model, the average runtime for each epoch is approximately 3.3 minutes hour for training and 0.5 minute for evaluating and testing.

The number of parameters for RoBERTa-base, CodeBERT-base and XLM-RoBERTa models are 125M, 125M and 270M respectively.

E Annotation Interface

We present the annotation interface used to collect the question, answer, and question type in Figure 10. Annotators can choose the messages corresponding to the question or answer text, and modify the texts in the interface. Annotators also select a question type for every question.

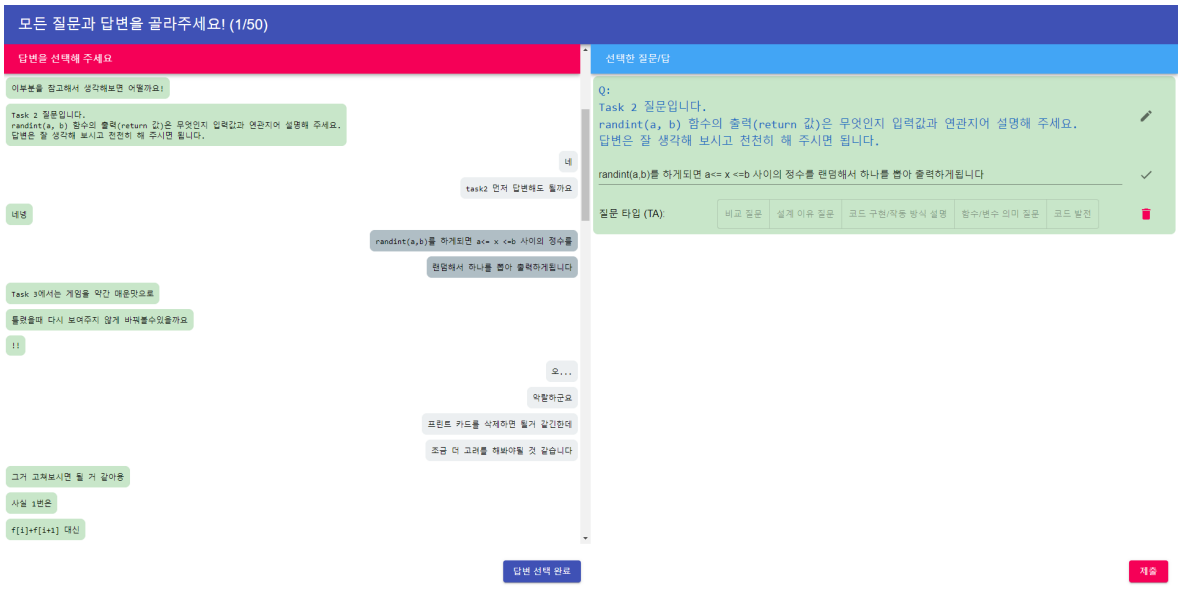

Figure 10: The annotation interface for question, answer and type selection. On the left, the chat log is presented. On the right, annotators can modify the question and answer texts and select the question type.

We present the annotation interface used to collect the code and the code span in Figure 11. Annotators choose the code for the question given, and select code spans with a code line as a unit.

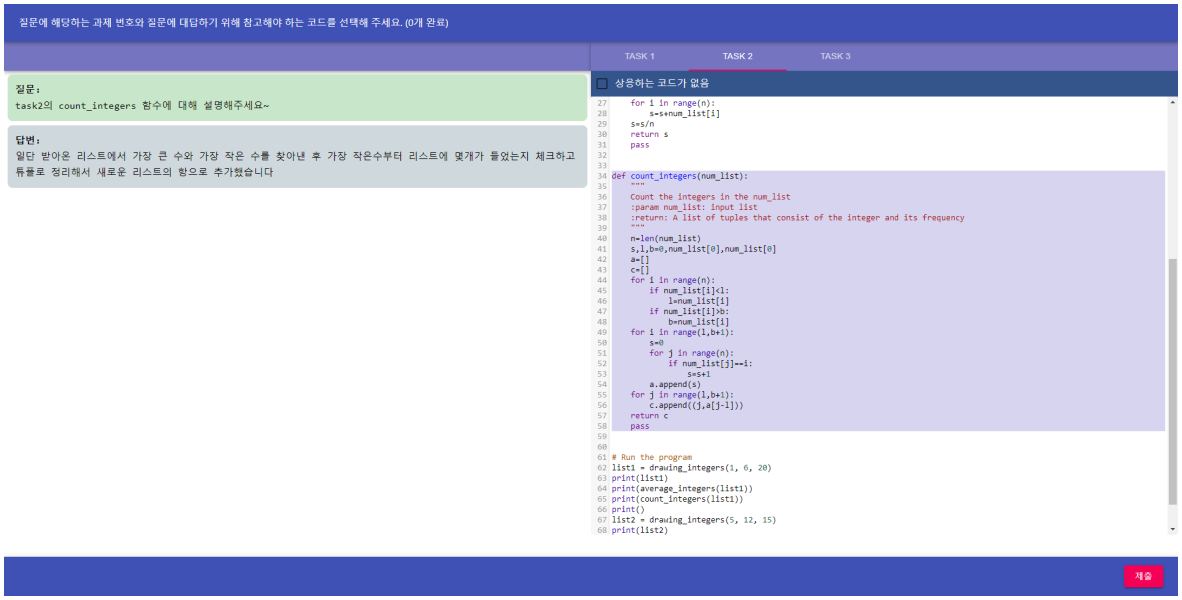

Figure 11: The annotation interface for code line selection. On the left, the question and answer texts are presented. On the right, annotators select the correct task for the given question and answer, and select the code lines that provide information to answer the question.

The full-text instructions for QA annotation can be found in this link. The instructions for code line annotaion can be found in this link.