Autograting “Explain in Plain English” questions using NLP

Max Fowler
University of Illinois
Urbana, IL, USA
mfowler5@illinois.edu

Binglin Chen
University of Illinois
Urbana, IL, USA
chen386@illinois.edu

Sushmita Azad
University of Illinois
Urbana, IL, USA
sazad2@illinois.edu

Matthew West
University of Illinois
Urbana, IL, USA
mwest@illinois.edu

Craig Zilles
University of Illinois
Urbana, IL, USA
zilles@illinois.edu

ABSTRACT
Previous research suggests that “Explain in Plain English” (EiPE) code reading activities could play an important role in the development of novice programmers, but EiPE questions aren’t heavily used in introductory programming courses because they (traditionally) required manual grading. We present what we believe to be the first automatic grader for EiPE questions and its deployment in a large-enrollment introductory programming course. Based on a set of questions deployed on a computer-based exam, we find that our implementation has an accuracy of 87–89%, which is similar in performance to course teaching assistants trained to perform this task and compares favorably to automatic short answer grading algorithms developed for other domains. In addition, we briefly characterize the kinds of answers that the current autograder fails to score correctly and the kinds of errors made by students.

CCS CONCEPTS
• Social and professional topics → CS1; • Computing methodologies → Natural language processing.

KEYWORDS
code reading, Explain in Plain English, ASAG, NLP, CS1

ACM Reference Format:

1 INTRODUCTION
The historic difficulty in learning to program [24] may be in part due to a pre-mature emphasis on code writing rather than code reading. For example, a recent theory of instruction for introductory programming [42] suggests that students should understand common programming patterns/idioms via code reading activities prior to being asked to compose those programming patterns/idioms to solve problems (i.e., writing complete routines). This theory is well motivated by the research literature (discussed in Section 2) which recognizes that (1) expert programmers use programming language features in idiomatic ways, (2) learning to program is as much about learning these idioms as it is about learning the syntactic elements themselves, and (3) activities like code tracing and code reading provide novices a means of developing programming skills with a lower cognitive load than code writing itself and, hence, should be employed to prepare students for code writing.

While code tracing and code writing activities are commonplace in introductory programming (CS 1) courses, code reading is not nearly as prominent. We could just ask our students to study large banks of programming examples, but many students require an incentive to get them to do assigned reading activities [14, 30]. One approach to gradable code reading activities proposed in the literature is “Explain in Plain English” (EiPE) questions (as shown in Figure 1) that ask students to describe in natural language (e.g., English) the high-level behavior of a short piece of code. While EiPE questions are well regarded by researchers, they aren’t in widespread instructional use, presumably due to the burden of manually grading them and the slow feedback provided to students. In light of the large enrollments in many CS 1 classes, we sought to develop a means to automatically grade EiPE questions. As such, this paper poses the following research question: Can natural language processing (NLP) technology be used to automatically grade EiPE questions at accuracy rates comparable to humans?

In this work, we describe our initial development of an NLP-based autograder for EiPE questions and its use in a large-enrollment university-level introductory programming course. In particular we: 1) describe what we believe to be the first published implementation of autograded EiPE questions; 2) characterize the accuracy of the implementation as a function of the amount of training data, with our 87–89% accuracy suggesting EiPE may be simpler than other kinds of automatic short-answer grading; 3) compare the accuracy of the EiPE autograder with trained (non-researcher) teaching assistants (TAs) and find that the EiPE autograder has similar accuracy.

2 RELATED WORK
2.1 “Hierarchy” of programming skills
Researchers theorize that there is a loose hierarchy of programming skills with code writing at the top of the hierarchy, and many programming students struggle with tasks lower in the hierarchy [17].
These skills span from understanding syntax (as the easiest), to code tracing (executing code in your head for one particular input), to code reading/explaining (abstracting the behavior of code across all inputs), to code writing (as the most complex) [17]. It has been shown that for a given piece of code, task difficulty generally increases as we move up the hierarchy (e.g., tracing a swap vs. reading/explaining a swap vs. writing a swap) [20, 39], and students’ mastery of the lower level skills is predictive of their code writing ability [8, 18, 20, 37]. In particular, Lopez et al. find that students’ performance on tracing and code reading questions account for 46% of the variance in their performance on code writing questions [20]. Lister et al. state that, while their data doesn’t support the idea of a strict hierarchy, “We found that students who cannot trace code usually cannot explain code, and also that students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code.” [18]

Whalley et al. argue that in order for a novice to write a particular piece of code, they must be able to comprehend that same piece of code and the knowledge and strategies within it [39]. In particular, programmers need to be able to understand code at the relational level (i.e., can summarize the code’s purpose) and not just the multistructural (i.e., line-by-line) level [39]. Longitudinal studies find that students who are unable to explain code relationally early in the semester have difficulty writing code later in the semester [8].

It has been proposed that novice instruction should focus more on code tracing and reading [1, 6, 7, 18, 27, 40]. Lister et al. state, “It is our view that novices only begin to improve their code writing ability via extensive practice in code writing when their tracing and explaining skills are strong enough to support a systematic approach to code writing…” [18]

In particular, code reading activities may be among the best activities for helping novices learn to use common programming idioms. One important difference from novice programmers is that experts can automatically ‘chunk’ multiple syntax elements and process them as one unit [6, 12, 15, 25, 41], which reduces their cognitive load [35]. These chunks (or schema in the cognitive load literature) are developed through repeated experiences that have identifiable features in common [23] and are learned more efficiently in lower cognitive load activities (i.e., code reading rather than code writing) [36]. When students had relevant schema available, Rist found that students could and did reason forward from plan to code [31].

While all published works involving EiPE questions have been on summative assessments, Corney et al. suggest that “the value of EiPE problems may therefore be more as formative assessment rather than summative assessment” [7]. The challenges to using EiPE in formative assessment are having incentives in place to get the students to do the assignments [14] and giving the students immediate feedback on their understanding [29].

## 2.2 Automatic short answer grading (ASAG)

Automatically grading EiPE questions falls into the category of automatic short answer grading (ASAG). ASAG is characterized
by five criteria [3]: (1) student must recall external knowledge, (2) natural language responses, (3) response length of a phrase to a few sentences, (4) grading emphasizes content rather than writing style, and (5) focused prompts. ASAG is challenging because there are many ways to express a correct answers in natural language.

While there has been an evolution of ASAG techniques [3], current competitive ASAG systems are all based on machine learning [11]. These use a variety of features, including lexical (e.g., bag-of-words), morphological (e.g., stem matches), semantic (e.g., latent semantic analysis), syntactic (e.g., part-of-speech tagging), and surface (e.g., word count). A wide variety of machine learning techniques have also been used [11]. Most recently, dialog based systems and intelligent tutoring systems [28, 32, 33] and end-to-end models have been used for ASAG [19, 43].

Much of the work on ASAG is focused on K-12, so there is relatively little ASAG work on Computer Science subject matter. The main application of ASAG on computing content is the “Texas dataset” [26], which features traditional short answer questions from a data structures course (e.g. “What is the role of a prototype program in problem solving?”) and no questions related to describing code. Many researchers have used this data set to evaluate novel ASAG algorithms [13, 21, 34], but none of these papers are computer science domain specific.

The current state of the art of ASAG is only modestly accurate. Four publicly-available ASAG datasets have multiple results reported [9, 10, 26] (see [11] for a detailed summary). Table 1 lists the best reported results for ASAG datasets (as of early 2020).

### Table 1: The strongest reported results on publicly-available ASAG datasets [11].

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Score type</th>
<th>Best score</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas [26]</td>
<td>r</td>
<td>0.630</td>
<td>[13]</td>
</tr>
<tr>
<td>ASAP-SAS [10]</td>
<td>QWK</td>
<td>0.791</td>
<td>[16]</td>
</tr>
<tr>
<td>Beetle UA 5-way [9]</td>
<td>micro F1</td>
<td>0.780</td>
<td>[22]</td>
</tr>
<tr>
<td>SciEntsBank UA 5-way [9]</td>
<td>F1</td>
<td>0.692</td>
<td>[22]</td>
</tr>
</tbody>
</table>

3. **AUTOGRADED EIPE QUESTIONS**

In line with research literature recommendations (Section 2.1), we developed our EIPE system primarily for use as a formative assessment (e.g., homework), as our primary goal was repeated practice on a vocabulary extracted from students’ responses. Its vocabulary is created by tokenizing all correct student responses and counting the later, more complicated questions, where we are concerned that some students might not be able to understand the relationship between the correct description with the provided code, the solution provides a visual explanation of how the code relates to the description, highlighting common programming idioms.

When placed on a weekly homework assignment, EIPE questions were grouped into pools of 8–12 questions. Each time a student attempts the question, they are shown a random selection from the question pool. To get full credit on the question, students have to answer a number of questions correctly. When they correctly answer the question, they are awarded points, but there is no penalty for incorrect answers. This approach makes the activity somewhat tolerant of an imperfect autograder, as any false negatives only delay (rather than prevent) students from getting their points.

Because there was no penalty for wrong answers, we discovered that a group of students were putting in “garbage” answers to get the system to show its example correct answers. We attempted to discourage this behavior in two ways: 1) we created a garbage filter that prevents the display of our correct answers when the student submission is not deemed to be an actual attempt to answer the question, and 2) we turned off text selection for the correct answers shown to make copying them require additional effort.

3.1 **The EIPE autograder**

The autograder that we report here is significantly simpler than the state of the art ASAG. Our autograder is a logistic regression model on bag-of-words and bigram (pairs of adjacent words) features. In order to minimize the impact of non-word symbols on feature extraction, students’ responses need to be preprocessed.

The first step of preprocessing is to remove quotes in the students’ responses. This helps to handle cases where students quoted a variable name or the entire response. The second step is to insert spaces before and after mathematical operators such as the addition operator “+” and all kinds of brackets. This ensures that all mathematical expressions will be spaced equivalently between operators and operands.

Each response is then split into sentences using the Python natural language toolkit nltk’s sentence tokenizer. Each sentence is further split into a list of tokens by nltk’s word tokenizer. The resulting tokens could be words, numbers, or math symbols. These tokens are then cast to lower case and passed to a spellchecker to correct misspelled words. At the end of the above processing, each student’s response has been converted into lists of tokens.

During training, we select a set of bag-of-words and bigrams for each question individually. For each response in the training set, we first identify the set of words and bigrams present (ignoring duplicates). We then count what fraction of the training set has each word and bigram. The 40% most frequent words and 20% most frequent bigrams are then used as the feature set.

Each response is converted to a feature vector of 0s and 1s, where 1 at a index indicates a particular word or bigram in the feature set is present in the response. These vectors are then fed into scikit-learn’s logistic regression to train a model. The model produces binary correct/incorrect judgments.

The autograder’s garbage filter is a rule-based algorithm relying on a vocabulary extracted from students’ responses. Its vocabulary is created by tokenizing all correct student responses and counting
the occurrences of each token. The 75% most frequent tokens were considered as the non-garbage vocabulary. The garbage filter categorizes as garbage any response with two or fewer tokens or with less than 60% of its tokens in the non-garbage vocabulary.

4 METHODS

Our EiPE autograder was developed for and deployed in an introductory CS course for non-technical majors at a large U.S. university. This large-enrollment course (capped at 600 students) introduces basic principles of programming in both Python and Excel to a population largely without any prior programming experience. In this paper, we report results from both the Fall 2019 and Spring 2020 semesters. The course is taken predominantly by freshmen (67%) and sophomores (21%) and approaches gender balance (Fall 2019: 246 women/355 men; Spring 2020 258 women/307 men).

The autograding EiPE questions were deployed both on homework assignments during the first half of the semester and on proctored computer-based exams in each semester. We present results drawn only from student responses on exams because we can be assured that those results represent serious attempts by the students; we observed students attempting to “game the system” on the low-stakes homework, as discussed in Section 3.1. In Fall 2019, the EiPE questions appeared on a mid-term exam in the 12th week and, in Spring 2020, the EiPE questions appeared on a mid-term exam in the 6th week of the course. This discrepancy is due to the technology still being refined during the Fall 2019 semester.

In both cases, we used the pool of EiPE questions deployed on the homework during the 5th week of the course. Students took their exams in a Computer-Based Testing Facility [44] over a three-day period. To minimize the potential for cheating on these asynchronous exams, we randomly assigned students questions from the question pool on an individual basis [5]. Four of the problems in the pool were not included on the exam because they were significantly easier or harder than the rest.

In Fall 2019, students were given three attempts to submit a correct answer (the exam is graded interactively). Because we found that these multiple attempts resulted in additional false positives [2], in Spring 2020 we gave students a single attempt to answer the question. The students in Fall 2019 submitted a total of 1,140 responses and the students in Spring 2020 submitted a total of 582 responses.

4.1 Training Data and Ground Truth

The autograders were initially trained with data from a series of surveys. Each survey asked participants to provide two correct responses and two plausible incorrect responses for each EiPE question. These surveys were completed by the course’s instructor, TAs, and a collection of upper-level CS students. Students were compensated with Amazon gift cards. These surveys resulted in approximately 100–200 responses per question. Survey data was manually reviewed by a research team member to perform any necessary re-categorization of the responses. Our binary scores corresponded to 0-4 (incorrect) and 5-6 (correct) on a previously validated scoring rubric [4].

Prior to deployment on the exam, two members of the research team manually labeled the Fall 2019 students’ homework responses to these questions and used that as additional training data to improve the grader. The AI graders deployed on the exam were trained with 500–600 labeled responses per question.

Once an exam was completed, two members of the research team familiar with the course content scored each student response as correct/incorrect independent of the AI grader’s score. Correct responses must be correct at the relational level (non line-by-line descriptions) and unambiguous. Responses were not penalized for spelling or grammar as long as the intent was clear. If the two scorers agreed on a response, this was considered the final ground truth. For any disagreements between the scorers, ground truth was established by a process of discussion and reconciliation between both scorers and a third research team member until consensus was reached. The inter-rater reliability of the two research team members’ initial scores was a Cohen’s kappa of 0.83 (“almost perfect” agreement [38]). Grades were manually adjusted for any students who were incorrectly denied credit by the autograder.

4.2 Training human graders

Because members of the research team have scored thousands of EiPE responses and spent tens of hours reconciling challenging responses and developing a shared understanding of how to handle corner-cases, our level of experience and interest in EiPE grading far exceeds that of the teaching assistants (TAs) who would perform manual grading in the context of a large-enrollment course. As such, we perform our comparison to manual grading using an ecologically valid population: the TAs for the Spring 2020 offering of the course in which the data was collected. The eight TAs that participated were all graduate students in computer science.

To investigate differences in experience, the eight TAs were split into two groups of four (“Trained TAs” and “Minimally-trained TAs”) for different training regimens. The TAs were unaware of this split. IRB approval and TA consent were obtained for using their grading results.

Trained TAs: The four TAs in the “Trained TAs” group were given a joint 30-minute training session by a member of the research group. The training consisted of explaining the scoring criteria (e.g., high-level, correct, and unambiguous) to them in detail and then having them score example responses drawn from the Fall 2019 data in multiple rounds. Each round, the TAs independently scored five responses. Discrepancies between any of the TAs scores and the research team’s ground truth scores were collectively discussed and resolved before continuing on to the next round. There were four such rounds; each round used responses from a different question per round. We assigned the three returning Fall 2019 TAs to Group A because they had some prior exposure to the EiPE questions during that offering of the course.

Minimally-trained TAs: The four TAs in the “Minimally-trained TAs” group were all new to the course. They were given only a 5-minute training session by the same member of the research group, during which they were only shown the scoring criteria. No examples were worked through or discussed.

TA grading: Following the training, all eight TAs scored a selection of EiPE student responses from the Spring 2020 exam, without access to either the ground truth or the model’s score. Each TA received the same 100 responses for manual grading. The responses were stratified to disproportionately sample responses that the
model struggled with; both False Negatives (FNs) and False Positives (FPs). All the available FNs were sampled, while FPs were sampled at about twice the rate of the True Positives (TPs) and True Negatives (TNs). The TA responses were reweighed from the sample back to the full population rates for each category. These results were compared to the model’s performance on the Spring 2020 exam.

5 RESULTS

On the Fall 2019 exam, 67% of the students answered their EiPE question correctly and the autograder achieved an accuracy of 89%, with a 12% False Positive (FP) rate and a 9% False Negative (FN) rate. In Spring 2020, 45% of the students answered their EiPE question correctly and the autograder achieved an accuracy of 87%, with a 15% FP rate and a 10% FN rate. The model used a different threshold for marking answers correct in Fall 2019 and Spring 2020. We found that running Fall 2019’s responses using Spring 2020’s threshold reduced accuracy from 89% to 88%.

Mean accuracies for the model, trained TAs, and minimally-trained TAs are plotted in Figure 2. We did not find a statistically significant difference between the model and TAs, regardless of training. The model did perform about as well as three of our trained TAs and outperformed all but one minimally trained TA. These results, however, are not statistically significant because of the small number of TAs (N = 4 in each case). The TAs’ individual accuracies are sorted and plotted in Figure 3. Inter-rater reliability, F1 score, and accuracy relative to the ground truth for each TA individually are presented in Table 2.

Table 3 shows these results in the same metrics reported by the best current results on publicly-available ASAG data sets shown in Table 1. While clearly not directly comparable due to different data sets, we note that our model is achieving higher accuracy in spite of the fact that it is much less sophisticated. As we discuss in Section 6, we believe that this indicates that EiPE autograding is a simpler task than ASAG in general.

To understand how much training data is needed to obtain a reasonable AI grader and whether there is a qualitative difference between survey data and student homework data, we trained graders with different subsamples of data and show the mean of the grader’s performance on the Fall 2019 responses in Figure 4. There are three sources of training data: (1) a subset of the survey data, (2) a subset of the student homework data, and (3) both, meaning all of the survey data and a subset of the student homework data. Although more data consistently led to better performance, the student homework data seems qualitatively better than survey data, suggesting that the course staff and senior students creating the survey data were only somewhat able to generate realistic training data.

5.1 Characterizing Model Failure

Discussion of False Negatives: The model primarily struggles with two types of questions. First, in binary conditionals (e.g., “Is a number even?”), the model struggles with responses like “whether the variable is even (True) or odd (False)”, and score them as correct.

Second, in bi-variate comparisons (“Max of two numbers”, “Is x a factor of y?”) the autograder seems to struggle with recognition of uncommonly used synonyms and syntax. For example, the answer “returns the largest variable” is incorrectly marked wrong, but for all such responses, when we change the word ‘variable’ to ‘number’, then the model scores them as correct. These misclassifications seem to be due to few correct responses in the training data that use the word ‘variable’. In addition, the model struggles with unnatural
We were genuinely surprised that our very simple model performed what “it” refers to. Minor typos, however, do not result in FNs. Discussion of False Positives: We see both a larger number of FPs (101 FPs vs 29 FNs) as well as more diversity in the types of questions and responses that caused them. In general, as long as the student response has the keyword or a set of crucial words that appear in the training set of correct responses and does not have the corresponding incorrect keyword (e.g. ‘odd’ when looking for ‘even’), the model tends to score it as correct. Extra information or word order leading to an incorrect meaning tend to be ignored. This is likely due to our model using a bigram (+ bag-of-words) approach. Bigrams are not able to capture negative signals when negative signals are more than two words long and the bigrams in that negative signal have also appeared in a correct answer. For this high stakes exam deployment, we biased the model to avoid false negatives [2]. While the FP rate at 12% is higher than we would like, it should decrease with model improvement.

6 DISCUSSION

We see both a larger number of FPs (101 FPs vs 29 FNs) as well as more diversity in the types of questions and responses that caused them. In general, as long as the student response has the keyword or a set of crucial words that appear in the training set of correct responses and does not have the corresponding incorrect keyword (e.g. ‘odd’ when looking for ‘even’), the model tends to score it as correct. Extra information or word order leading to an incorrect meaning tend to be ignored. This is likely due to our model using a bigram (+ bag-of-words) approach. Bigrams are not able to capture negative signals when negative signals are more than two words long and the bigrams in that negative signal have also appeared in a correct answer. For this high stakes exam deployment, we biased the model to avoid false negatives [2]. While the FP rate at 12% is higher than we would like, it should decrease with model improvement.

The data in Figure 4 supports the assertion that the current model is not limited by the amount of training data. The model’s accuracy is almost constant once we reached 350 items in the training set. Furthermore, it is not surprising that the student homework responses were more effective than survey data for training the algorithm to predict student exam responses. The surveys did enable us to deploy the algorithm in the low stakes homework context to collect that homework training data, but our conclusion is that we could get by with fewer survey responses, especially if we were to quickly score early homework responses and re-train the model.

Our current implementation’s accuracy is in the same ballpark as our average course TAs, even after training. We believe that even just using the state of the art ASAG techniques should enable an algorithm to be competitive with even our most accurate TAs.

7 CONCLUSION

We believe this paper reports on the first instance of autograding “Explain in Plain English” (EiPE) questions. It appears that autograding EiPE questions is easier than the more general problem of automatic short answer grading. By training a simple model (bag-of-words + bigrams) using student homework responses to EiPE questions, we were able to achieve similar accuracy in autograding EiPE questions to manual grading by course TAs.

We believe that autograded EiPE questions have the potential to significantly impact instruction in introductory programming courses. Many researchers have advocated for a greater emphasis on code reading activities, and automated EiPE activities can effectively scale to even large enrollment CS 1 courses.

There are many directions for future work. First, recent advances in ASAG and NLP more broadly should be introduced into the model. Second, we suggest evaluating EiPE autograding on a broader range of questions to better characterize how autograding accuracy varies across questions and populations. Such an analysis might provide insight into how to further improve the algorithm. Finally, once a mature autograding EiPE system is available, it will be important to identify best practices for adopting it into existing courses and measuring the impact a larger emphasis on code reading has on introductory programming instruction.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant numbers DUE-1347722 and DUE-1915257.

Figure 4: The performance of the AI grader on the 1,140 exam responses from Fall 2019, when trained on different combinations of data with different sample sizes. Error bars are 95% confidence intervals.