Evaluating Large Language Model Code Generation as an Autograding Mechanism for "Explain in Plain English" Questions

David H. Smith IV
University of Illinois
Urbana, IL, USA
dhsmith2@illinois.edu

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

ABSTRACT

The ability of students to "Explain in Plain English" (EiPE) the purpose of code is a critical skill for students in introductory programming courses to develop. EiPE questions serve as both a mechanism for students to develop and demonstrate code comprehension skills. However, evaluating this skill has been challenging as manual grading is time consuming and not easily automated. The process of constructing a prompt for the purposes of code generation for a Large Language Model, such OpenAI's GPT-4, bears a striking resemblance to constructing EiPE responses. In this paper, we explore the potential of using test cases run on code generated by GPT-4 from students' EiPE responses as a grading mechanism for EiPE questions. We applied this proposed grading method to a corpus of EiPE responses collected from past exams, then measured agreement between the results of this grading method and human graders. Overall, we find moderate agreement between the human raters and the results of the unit tests run on the generated code. This appears to be attributable to GPT-4's code generation being more lenient than human graders on low-level descriptions of code.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

GPT-4, Large Language Models, EiPE, Autograding

ACM Reference Format:
David H. Smith IV and Craig Zilles. 2024. Evaluating Large Language Model Code Generation as an Autograding Mechanism for "Explain in Plain English" Questions. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626253.3635542

1 INTRODUCTION AND BACKGROUND

The emergence and widespread popularity of services powered by "Large Language Models" (LLMs) has sparked concerns amongst educators. These concerns appear to primarily stem from the ease with which these models can be used to generate solutions to queries from a diverse set of topics [2]. Despite these concerns, there is equal excitement for the potential revolution in education this technology might bring about, particularly in the domain of computer science education [3]. With the rapid adoption of LLM powered tools like GitHub Copilot and ChatGPT, success as a programmer may increasingly hinge on one's ability to effectively employ these tools and evaluate the results they produce.

One such area of computer science education that may benefit from LLMs is the automatic grading of "Explain in Plain English" (EiPE) questions. In these questions, students are shown a segment of code and asked to demonstrate their comprehension of that code by providing a high-level description of its functionality [7]. EiPE questions typically require short answer responses their grading has often been manual which limits the ability for students to practice and receive timely feedback. Existing EiPE autograders take steps toward addressing this by giving students immediate feedback on the correctness of a response [4]. Though these graders perform similarly to a trained teaching assistant, to develop each EiPE question a large quantity of responses must be labeled which leads to overhead in the question authoring process.

To address this these limitations, we propose "Code Generation Based Grading" (CGBG). This grading approach uses a student's EiPE response to generate code via an LLM. The generated code is then tested for correctness through unit tests to determine if a student's description of a segment of code was able to generate functionally equivalant code. In this way, CGBG enables feedback by providing students the generated code and the results of the unit tests. It also simplifies the question authoring process by eliminating the need for human data labeling and replacing it with the simple authoring of unit tests. As an initial step towards evaluating CGBG's effectiveness as an EiPE grading mechanism we evaluate several implementations of CGBG via the following research questions:

RQ1 What is the agreement between trained human raters and code generation based grading?
RQ2 What relationships exist between the features of a given question and the agreement on that question?

2 CODE GENERATION BASED GRADING

The "Code Generation Based Grading" (CGBG) process we propose is divided into three distinct steps [8]. First, a student's response to an EiPE question and a pre-prompt instructing the LLM to generate a function based on the description are combined. This combined prompt is used to generate a function via GPT-4 which is then run against a set of manually defined unit tests. The student's grades for their response is then dependent on the success of the unit tests.

This process simplifies the question authoring process for producing automatically graded EiPE questions to be more similar to that of automatically graded code writing questions. This authoring process is simpler than prior EiPE autograders, which require the creation of human labeled datasets and training of NLP models [1]. This also offers the benefit of allowing the instructor to concretely define what aspects of the code and edge cases should be described in the prompt through the unit tests.

However, a major consideration with this grading approach is how to account for the non-deterministic nature of GPT-4's responses. Through the model's temperature parameter, the "creativity" of the model can be controlled. A prompt made with a temperature of zero will always produce the same response. By increasing the temperature value the responses will gradually become less deterministic and more creative with the model's interpretation of the prompt. For the purposes of grading, we want to account for the possibility of a student being marked incorrect not because their prompt was poorly formed but simply because the model generated an overly creative response. With this in mind, we compare the following three grading approaches:

- **Single Response** A single response at 0.0 temperature is used for grading.
- **Best of 5:** Five responses are generated at 0.5 temperature. A prompt is graded as correct if at least one response passes all unit tests.
- **Majority Vote:** Five responses are generated at 0.5 temperature. A prompt is graded as correct if at least 3 of the 5 responses pass all unit tests.

3 METHODS

To evaluate the alignment between human graders and each of the proposed CGBG approaches, we conduct an analysis using historical data from a large introductory Python course taught at a large university in the United States. EiPE questions are an integral part of the course with students receiving explicit instruction on how to form adequate responses. When these questions appear on exams, they are graded manually by two trained teaching assistants (TAs). These TAs independently assessed responses using a rubric which covered which evaluated responses on three dimensions: 1) the functional correctness of the student's description, 2) a lack of ambiguity in the description, and 3) the description providing a high-level description of the codes purpose rather than a "line-by-line" description. After grading the responses the TAs met to consolidate their grades and reconcile any differences. A total of 6380 responses to 42 EiPE questions from past exams were subject to each of the three CGBG approaches. Cohen's $\varkappa$ was used to measure agreement between the human graders and each of the CGBG approaches across the 42 questions [6].

4 RESULTS AND DISCUSSION

The CGBG approaches introduced in this paper all achieve a moderate agreement with human raters with very little difference between them (Figure 1). Closely analyzing which questions fell into categories of high ($\kappa \geq 0.6$), medium ($0.4 < \kappa < 0.6$), and low ($\kappa < 0.4$) agreement revealed the primarily limitation of this approach is its inability to distinguish between high and low level descriptions. This limitation is most prominent on EiPE questions that require students to describe a short segment of code (1-2 lines). For example, consider the following segment of code:

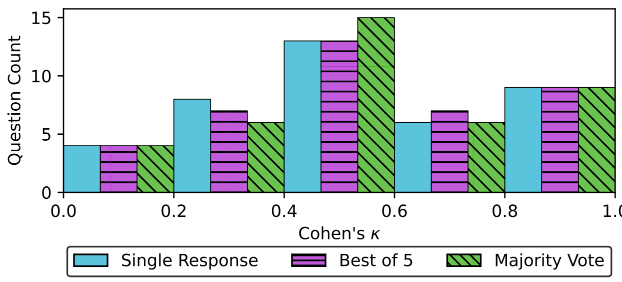

Figure 1: The distribution of agreements between each CGBG approach and human graders for each of the EiPE questions being evaluated.

```python
def foo(x):
    return x % 2 == 0
```

A low level description might be "*determines if x mod 2 equals 0*" whereas a high level description would be "*determines if x is even*". As such, CGBG might be best suited for cases where instructors don't place as strong an emphasis on obtaining high-level descriptions. It may also be well suited for longer, more complex segments of code where heuristics such as limiting the length of students' responses could be used to nudge students towards higher level descriptions of the code.

There are several affordances this grading approach provides. Historically, deploying EiPE questions and evaluating their responses has either involved manual grading or the training of a model for each question. The use of CGBG reduces the authoring process to creating a sample code segment and a series of test cases. EiPE autograders have also faced challenges relating to a lack of transparency in their underlying grading mechanisms which can cause students to distrust the results they produce [5]. The proposed CGBG approaches aim to address this issue by providing students with the generated code as feedback along with the results of tests cases run on that same code. Future work will explore the impact of this feedback on students' ability to correct short coming in their responses and improve subsequent submissions.

REFERENCES

[1] Sushmita Azad. 2020. *Lessons learnt developing and deploying grading mechanisms for EiPE code-reading questions in CS1 classes.* Ph. D. Dissertation.
[2] Debby RE Cotton, Peter A Cotton, and J Reuben Shipway. 2023. Chatting and cheating: Ensuring academic integrity in the era of ChatGPT. *Innovations in Education and Teaching International* (2023), 1–12.
[3] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2022. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. *arXiv preprint arXiv:2210.15157* (2022).
[4] Max Fowler, Binglin Chen, Sushmita Azad, Matthew West, and Craig Zilles. 2021. Autograding" Explain in Plain English" questions using NLP. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.* 1163–1169.
[5] Silas Hsu, Tiffany Wenting Li, Zhilin Zhang, Max Fowler, Craig Zilles, and Karrie Karahalios. 2021. Attitudes surrounding an imperfect AI autograder. In *Proceedings of the 2021 CHI conference on human factors in computing systems.* 1–15.
[6] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.
[7] Laurie Murphy, Renée McCauley, and Sue Fitzgerald. 2012. 'Explain in plain English'questions: implications for teaching. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education.* 385–390.
[8] David H Smith IV and Craig Zilles. 2023. Code Generation Based Grading: Evaluating an Auto-grading Mechanism for" Explain-in-Plain-English" Questions. *arXiv preprint arXiv:2311.14903* (2023).