Evaluating Correctness of Student Code Explanations: Challenges and Solutions[⋆]

Arun-Balajiee Lekshmi-Narayanan[1,*], Peter Brusilvosky

Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA

Abstract

Educational data used by data mining approaches in the domain of Computer Science education primarily focused on working with code-based data, such as student homework submissions. However, the increased use of natural language techniques and Large Language models (LLM) in all domains of learning including Computer Science education is now producing an abundance of natural language data, such as code explanations generated by students and LLMs as well as feedback and hints produced by instructors, TAs, and LLMs. These data represent new challenges for CSEDM research and need new creative approaches to leverage. In this paper, we present a first attempt to analyze one type of these new data, student explanations of worked code examples. The main challenge in working with these data is to evaluate the correctness of self-explanations. Using a dataset of student explanations collected in our previous work, we demonstrate the difficulty of this problem and discuss a possible way to solve it.

Keywords

code explanations, worked examples, automated assessment

1. Introduction

The majority of the work in computer science educational data mining (CSEDM) relied so far on datasets that collected traces of learner work with various learning content or datasets with student submissions to programming assignments [1] Using these datasets, researchers were able to explore a range of novel approaches including finding knowledge components [1], debugging [2, 3], and detecting cheating [4]. However, as newer types of dataset become openly available for analysis, new methods need to be developed to leverage this data [2].

With recent research on student self-explanation of code fragments [5] as well as the use of LLMs and students to generate code explanations automatically [6], an increasing number of datasets contain free–form code explanations. In this work, we consider one such dataset with code explanations generated by students and instructors [7]. This dataset was annotated to mark the correctness of each student's explanation and to assess the similarity between students and instructors' explanations for the same code lines. The goal we want to achieve by working with this dataset is to distinguish correct and incorect explanations. This goal has practical value. An approach that could reliably identify incorrect explanations could be used to build an intelligent tutor to support the self-explanation process [5].

Starting with a review of relevant work, the paper discusses several approaches to distinguish correct and incorect explanations. Since our dataset contains "ground truth", i.e., human expert annotation of each explanations as as correct or incorrect (including inter–rater reliability) we are able to use the dataset to illustrate the feasibility of these approaches.

More specifically, the remaining part of the paper focuses on two groups of approaches:

1. *Use surface-level features*: this group of approaches use "surface–level" lexical and readability features that could be easily extracted from the text of student or expert explanations. This is discussed further in Section 3.1.
2. *Use expert explanations*: this group of approaches attempts to calculate various similarity metrics between student explanations and expert explanation and use the obtained similarity to distinguish correct and incorect explanations. This is described further in Section 3.2.

2. Related Work

Corpora for free-form student answers such as reflective essays [8] and argumentative writing [9], provide interesting examples of use cases that are different from traditional log data. The ability to analyze this data is important to provide feedback when assessing students' free–form responses. Tools, such as COH–METRIX [10] and EDU-Convokit [11], offer several options to analyze textual educational data; however, our dataset of free–form code explanations needs slightly different methods to evaluate correctness.

Some examples in the natural language processing domain offer encouraging examples of using surface-level features to models for various tasks. Schwartz and colleagues explore surface features such as the word or character n–gram and length of sentences to build a classifier to identify author writing styles in a CLOZE story task [12]. Some examples in the case of language inference tasks consider word-level similarity–based approaches [13]. When constructing adversarial examples for natural language inference tasks, another work considers surface–level cues such as words that contradict or are negative ("not"). Negative sampling is another approach that uses surface–level features to construct synthetic examples that can help build robust classifiers [14].

Li and colleagues [15] discuss the student perceptions on the potential errors autograder may provide as feedback to their submissions. This emphasizes the need to develop

CSEDM'24: 8th Educational Data Mining in Computer Science Education (CSEDM) Workshop, June 14, 2024, Atlanta, GA

[⋆]You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

[*]Corresponding author.

arl122@pitt.edu (A. Lekshmi-Narayanan); peterb@pitt.edu (P. Brusilvosky)
https://a2un.github.io (A. Lekshmi-Narayanan); https://sites.pitt.edu/~peterb/ (P. Brusilvosky)
0000-0002-7735-5008 (A. Lekshmi-Narayanan); 0000-0002-1902-1464 (P. Brusilvosky)

[1]https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=3458
[2]https://the-learning-agency.com/learning-engineering-hub/build/

better automated assessment techniques, for newer kinds of data such as code explanations discussed in our work. The earlier work of the same team [16] discuss the use of an autograding system that evaluates student explanations to code in plain English. They show the potential limitations of using finetuned AI models for autograding accuracy by comparing with TAs at different levels of grading expertise and do not find statistically significant results – either owing to sample size or the AI model actually not performing better than TAs at the task. This necessatitates the possibilities to explore better finetuned AI models that have higher accuracy, with lower rates of false positives and negatives. In this work, we consider a context quite similar to their work, however, we use student self-explanations produced as a part of the learning process rather then explanations produced for grading. We also start from scratch by identifying features in student explanations to classify them as correct or incorrect. Like this work, we are observing that the use of surface–level linguistic features may not help in differentiating student explanations by correctness. Possible extensions would involve the use of contextual emebeddings and LLMs which we also currently explore as a follow-up to this current work–in–progress.

Denny and colleagues [17] explore student code explanations in plain English in a different context. In this work, code explanations are used to encourage students to think deeply about the problem so that using their code explanations LLM can generate a code equivalent to the code the student is trying to explain. Additionally, they evaluate the student explanations progress up the classifications of SOLO taxonomy. They also conduct a user study to evaluate students' perceptions on traditional approaches such as code–writing in comparison to approahces like code explanations.

Haller and colleagues [18] survey automated assessments tools that are used to evaluate short answer essays. They discuss hand–engineered appraoches in combination SVM or KNN based classifiers. In our case, the goal is not to build best classifiers for the task, but to evaluate if the features themselves reveal differences between correct and incorrect student explanations.

Lin and colleagues [19] explore the use of a finet–tuned GPT model that can be used to provide personalized, adaptive feedback to students. They use new metric that is an extension to the precision / recall –based Intersection-Over-Union metric to evluate the LLM–based feedback and compare with human feedback in a user study. For current work–in–progress this idea is the next target to achieve in the context to code explanations of worked examples in programming.

Leinonen and colleagues [6] compare ChatGPT-generated explanations with student explanations. In our work, we are interested in classifying student explanations as correct or incorrect. In an ongoing extension to this work, we also focus on using ChatGPT–based interventions to solve this challenging problem.

3. Method

Inspired by previous work [20], we extract surface-level features from student and expert explanations alone and generate pairwise similarity scores between student and expert explanations for the same code lines. This data is applied to evaluate the correctness of the student's explanation for a given line of code.

3.1. Surface Features

We try to assess the correctness of student explanations using the following easily extracted features.

1. *Explanation Length* is calculated as the number of words to check that longer student explanations are correct. This is a useful metric for tasks such as persuasive essay evaluation [9] and we expect this could work for assessing code explanations also.
2. *Lexical Density* is calculated by the ratio of the number of nouns, adjectives, verbs, and adverbs (tagged in the sentence using a Spacy POS Tagger [3]) over the overall number of words in the sentence (Ure LD formula [4]). We expect that correct student explanations are lexically denser.
3. *Gunning Fog Readability* is the metric to evaluate the grade level to understand a text. We hypothesise that the correct student explanations might have higher scores (require more technical knowledge to understand) than incorrect explanations.

3.2. Expert–Student Similarity Features

We consider the pairwise similarity between expert and student explanations to assess correctness of student explanations. Following our previous work [20], METEOR [21], BERTScore [22] and chrF [23] are considered to evaluate the pairwise similarity between student and expert explanations for a given line of code. We expect correct student explanations to be more similar to expert explanations than incorrect explanations. We choose this combination of metrics because METEOR and chrF scores measure character and token level similarities, while BERTScore estimates semantic similarity by using cosine similarities between the contextual word embeddings of the two explanations. The similarity scores are between 0 and 1.

4. Dataset

We use a dataset of line–by–line explanations provided by students in an study in which they were asked to explain worked examples [24, 7]. The study included four Java worked code examples: some basic examples focused on array search and print statements and more difficult examples focused on object-oriented principles. Among about all expert explanations in the dataset, we considered upto 2 expert explanations for every line of code. In the original dataset, the majority of the student explanations were provided in a single sentence; however, a fraction of explanations included two or more sentences. For the purpose of this study, we excluded these multi-sentence explanations retaining between 23 and 26 single-sentence student explanations per line of code. The key datset parameters are shown in Table 1 and sample explanations are provided in Figure 1 (metadata columns are omitted). There is a known imbalance in the dataset between correct and incorrect examples (1234 instances of single sentence student explanations annotated as correct and 70 instances of single sentence student explanations annotated as incorrect). We calculated the average

[3]https://spacy.io/usage/linguistic-features
[4]https://en.wikipedia.org/wiki/Lexical_density

Dataset Property	Value
# Single Sentence Student–Expert (All Experts) Pairs	1854
# All Sentence Student–Expert (All Experts) Pairs	3019
# All Sentence Student–Expert (All Experts) Pairs Annotation Agreement	88.24%
# Worked code examples	4
# Lines per example	≈ 8
# Single Sentence Student–Expert (Expert 1 & 2) Pairs	1304
# Student–Expert (1 & 2) Explanation Pairs with Student Correct	1234
# Student–Expert (1 & 2) Explanation Pairs with Student Incorrect	70

Table 1
A summary of the properties our dataset.

percentage agreement for the annotation of correctness all sentence all experts student pairs of explanations (see Table 1). More details on the dataset is available in our past work [7].

Program: PointTester.java **Line number:** 14 **Line code:** `x += dx;`
Expert1: To shift the x-coordinate of the point, we need to add dx to the value of the x-coordinate of the point.
Student1: move the x coord the amount that the argument specified
Student2: Adds the first inputted value to X.
Student3: increases the value of x by the amount of the first parameter in the function.
...
Student23: The value of dx is added to variable x.

Figure 1: A slice of the dataset showing a subset of expert and student explanations for the same line of code.

5. Results

In this section, we group the results by the surface–level metrics used to evaluate the correctness of student explanations and similarity–based metrics where pairs of student and expert explanations were used. While extensive statistical results could be performed (such as t-tests to compare means), we preferred to perform exploratory analysis before digging deeper with our analysis.

5.1. Lexical Based Surface Metrics

5.1.1. Explanation Length

Using the length of the explanation, we observe if the expert and student explanations can be distinguished, we observe that the explanations marked correct have lengths of different words. Some lengths for correct explanations are the same as those for incorrect explanations (see Figure 2a). This may be because the student explanations are generally similar in length, regardless of whether they are annotated as correct or incorrect.

5.1.2. Readability Metrics

We observe that it is impossible to differentiate correct from incorrect student explanations using lexical surface metrics(see Figure 2b). This may be because the student explanations are not technically different in their explanation levels but the concepts in computing that are used to explain the line of code, which we observed when annotating the dataset.

5.1.3. Lexical Density

We observe that the lexical density also may not differentiate good and bad student explanations. (see Figure 2c). The lexical density measures a more linguistic aspect of the explanations by the parts of speech, which may not necessarily evaluate the conceptual aspects of the explanations. This is because the concepts may not be associated with a particular kind of speech and are more connected with the ontology of concepts in computing.

5.1.4. Vocabulary

Correctness also does not seem to depend on the vocabulary of the student explanations (see Figure 2d). The vocabulary in the sentence is more of a linguistic measure. Hence, this may not necessarily capture the conceptual ontology in computing such as those discussed in an earlier work of JAVA Parser [25].

5.2. Expert–Student Explanation Similarity

5.2.1. ChrF score

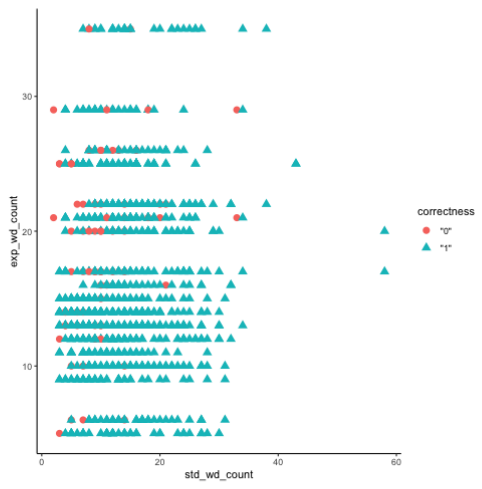
We observe that the differences considering the class imbalance between the correct and incorrect explanations could create an issue with the threshold to differentiate the explanations using this score (see Figure 3a). Further inspection of similarity scores at a line–by–line level shows that irrespective of the expert explanation that is used to calculate the similarity, the correct and incorrect student explanations cannot be separated easily by their ChrF score (see Figure 4a).

5.2.2. METEOR Metric

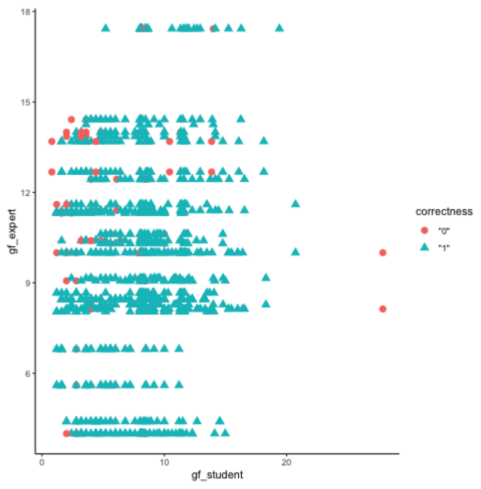
There is a more noticeable difference in the METEOR similarity scores between the correct and incorrect student explanations. This could be due to n-gram level word alignment. The density plots of the METEOR similarity scores distribution show that most incorrect explanations have a METEOR score below 0.3, as shown in Figure 3b. However, more than 50% of the correct explanations also have a METEOR similarity score below 0.3. Thus, irrespective of the expert explanation that is used to calculate the similarity, the correct and incorrect student explanations cannot be easily separated using the METEOR score (see Figure 4b).

5.2.3. BERTScore

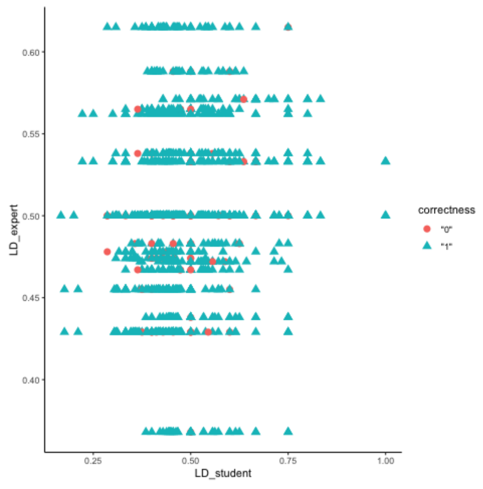
While we expected better performance of BERTScore in separation of correct and incorrect explanations, the density plots of the BERTScore distribution for correct and incorrect explanations show very little differences. We may



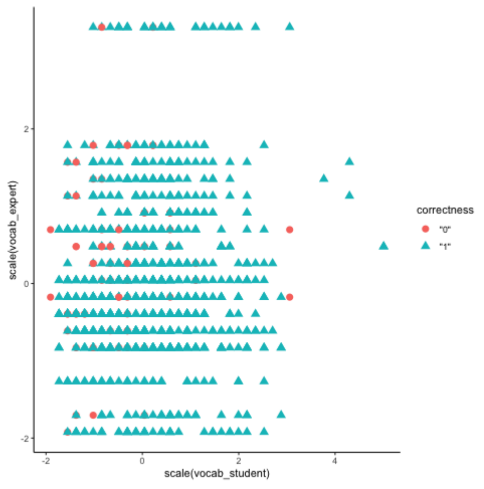
(a) Student and Expert Explanation Length



(b) Student and Expert Explanation Gunning-Fog



(c) Student and Expert Explanation Lexical Diversity



(d) Student and Expert Explanation Vocabulary

Figure 2: Scatter plots of various text linguistic ("surface") metrics. The x and y represent the student and expert sentence values, respectively. The colors, shapes, and sizes represent the annotation of the students' explanations for their correctness or incorrectness. Correct and incorrect student explanations are not differentiable by the lexical surface metrics.

Similarity Metric	**Incorrect** (Mean,SD)	**Correct** (Mean,SD)
chrF	0.305, 0.114	0.361, 0.140
METEOR	0.140, 0.091	0.283, 0.170
BERTScore	0.874, 0.028	0.894, 0.024

Table 2
The mean similarity scores between expert and student explanations are not different for the student explanations annotated as correct from incorrect. The most difference is observable with the METEOR metric, also observed with the plots

observe differences if we pre-trained a RoBERTa model over instances from the dataset. (See Figure 3c). As before, the inspection of the similarity scores at a line–by-line level shows that regardless of the expert explanation that is used to calculate similarity, the correct and incorrect student explanations cannot be separated easily (see Figure 4c).

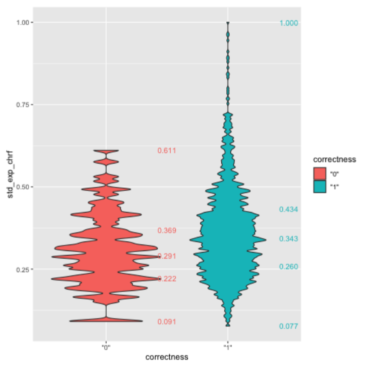
5.2.4. Similarity Correlations

When drawing similarity scores between the student and expert explanations, we can take the average similarity of the student explanation per line of code per solution with the two expert explanations. This, we can calculate this for all the lines of all the programs and observe that the while the range of values of the three similarity scoring metrics are different, they are highly correlated ($p < 1e-6$, $0.5 \leq corr \leq 0.6$), as shown in Figures 5a and 5b.

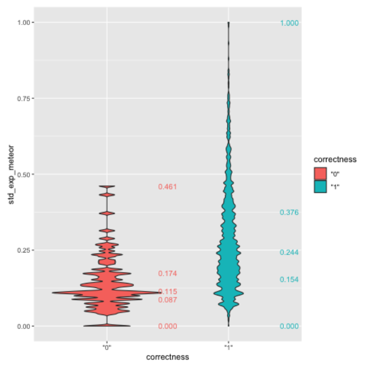
6. Conclusion

In this work, we present the challenges of analyzing new kinds of datasets such as the code explanations dataset in this paper. We observe that we need more sophisticated metrics to evaluate student explanations as "good" or "bad" and surface-level metrics are mostly ineffective in evaluating student explanation correctness. We present similarity-based metrics also not performing well in separating the "good" from the "bad" student explanations.

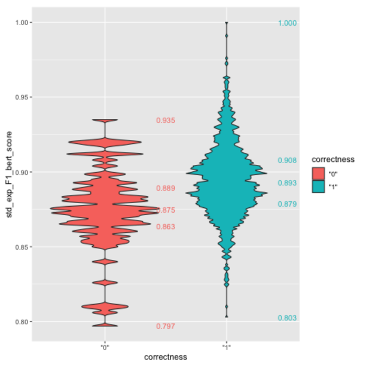
Our work has several limitations. We did not consider the use of combinations of lexical and similarity-based features to classify student explanation correctness. The goal of this paper is not to present the best possible classifier, rather to show the difficulty in identifying useful features



(a) Character F Similarity

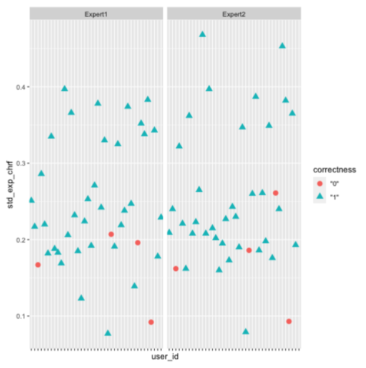


(b) Meteor Similarity

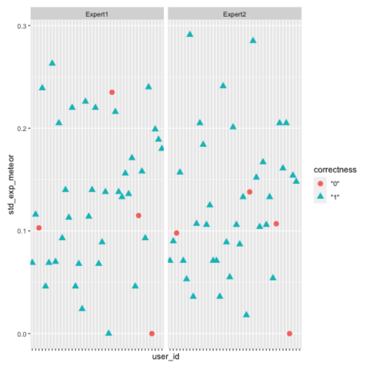


(c) BERTScore Similarity

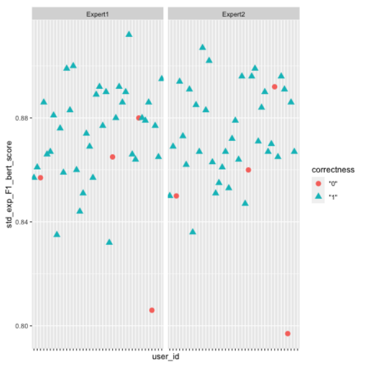
Figure 3: In the Figures 3a, 3b, 3c, we observe that student and expert explanations are similar irrespective of whether the student explanations are annotated as correct or incorrect.



(a) Character F Similarity



(b) Meteor Similarity



(c) BERTScore Similarity

Figure 4: The scatter plots for the 3 similarity metrics used between expert and student explanations for a given line of code in a given program separated by correct vs incorrect presents some differences in scale across the different similarity metrics (refer Figures 4a, 4b, 4c)

.

to differentiate correct from incorrect student explanations. We are addressing this in our ongoing work, with the use of LLMs to assess the correctness and provide feedback to student explanations. Our expert explanations may not have sufficient and diverse correct (positive) and incorrect (negative) examples to build robust classifiers. We are developing cross–validation techniques to build better classifiers that are exposed to various synthetic and real–world examples to evaluate student explanation correctness. In this work, we did not present similar results by considering multiple sentences for both student and expert explanations. While this is important, we chose to present in this work a prototype for single sentence evaluation which is scalable with aggregation techniques, which we will be presenting in an upcoming future work. This dataset does not cover the cases where students improve over time with providing correct explanations to lines of code as they progress through harder programming solutions. We will explore this in a longitudinal study as we build a system which will have the option to present harder examples for students to explain as they are evaluted correct with a better classifier that utilizes several of the current analyses presented in this work as evidence.

Acknowledgments

We thank Jeevan Chapagain for his efforts in the annotation of the dataset. We thank Dr. Xiang Lorraine Li for her contributions to writing this draft and for the natural language processing topics of the paper.

References

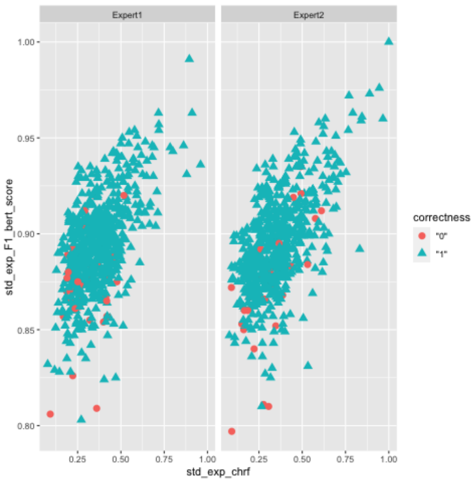
[1] Y. Shi, R. Schmucker, M. Chi, T. Barnes, T. Price, Kc-finder: Automated knowledge component discovery for programming problems., International Educational Data Mining Society (2023).

[2] A. M. Kazerouni, R. S. Mansur, S. H. Edwards, C. A. Shaffer, Student debugging practices and their relationships with project outcomes, in: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 2019, pp. 1263–1263.

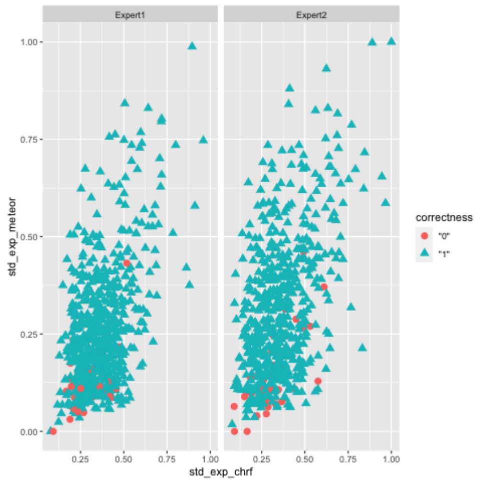
[3] P. Denny, J. Prather, B. A. Becker, Error message readability and novice debugging performance, in: Proceedings of the 2020 ACM conference on innovation and technology in computer science education, 2020, pp. 480–486.

[4] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, T. Price, B. Akram, Detecting chatgpt-generated code submissions in a cs1 course using machine learning models, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, 2024, pp. 526–532.

[5] P. Oli, R. Banjade, A. B. Lekshmi Narayanan, J. Cha-



(a) BERTScore ChrF Correlation



(b) METEOR ChrF Correlation

Figure 5: When we draw correlations, we observe that the three similiarity metrics are correlated to one another (refer 5a and 5b).

pagain, L. J. Tamang, P. Brusilovsky, V. Rus, Improving code comprehension through scaffolded self-explanations, in: Proceedings of 24th International Conference on Artificial Intelligence in Education, Part 2, Springer, 2023, pp. 478–483. URL: ttps://doi.org/10.1007/978-3-031-36272-9_75.

[6] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, A. Hellas, Comparing Code Explanations Created by Students and Large Language Models, 2023. arXiv:2304.03938, arXiv:2304.03938.

[7] A.-B. Lekshmi-Narayanan, J. Chapagain, P. Brusilovsky, V. Rus, SelfCode 2.0: Annotated Corpus of Student Self- Explanations to Introductory JAVA Programs in Computer Science, 2024. URL: https://doi.org/10.5281/zenodo.10912669. doi:10.5281/zenodo.10912669.

[8] X. Fan, W. Luo, M. Menekse, D. Litman, J. Wang, Coursemirror: Enhancing large classroom instructor-student interactions via mobile interfaces and natural language processing, in: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, 2015, pp. 1473–1478.

[9] S. A. Crossley, P. Baffour, Y. Tian, A. Picou, M. Benner, U. Boser, The persuasive essays for rating, selecting, and understanding argumentative and discourse elements (persuade) corpus 1.0, Assessing Writing 54 (2022) 100667.

[10] A. C. Graesser, D. S. McNamara, M. M. Louwerse, Z. Cai, Coh-metrix: Analysis of text on cohesion and language, Behavior research methods, instruments, & computers 36 (2004) 193–202.

[11] R. E. Wang, D. Demszky, Edu-convokit: An open-source library for education conversation data, arXiv preprint arXiv:2402.05111 (2024).

[12] R. Schwartz, M. Sap, I. Konstas, L. Zilles, Y. Choi, N. A. Smith, The effect of different writing tasks on linguistic style: A case study of the roc story cloze task, arXiv preprint arXiv:1702.01841 (2017).

[13] M. Glockner, V. Shwartz, Y. Goldberg, Breaking nli systems with sentences that require simple lexical inferences, arXiv preprint arXiv:1805.02266 (2018).

[14] Y. Belinkov, A. Poliak, S. M. Shieber, B. Van Durme, A. M. Rush, Don't take the premise for granted: Mitigating artifacts in natural language inference, arXiv preprint arXiv:1907.04380 (2019).

[15] T. W. Li, S. Hsu, M. Fowler, Z. Zhang, C. Zilles, K. Karahalios, Am i wrong, or is the autograder wrong? effects of ai grading mistakes on learning, in: Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1, 2023, pp. 159–176.

[16] M. Fowler, B. Chen, S. Azad, M. West, C. Zilles, Autograding "explain in plain english" questions using nlp, in: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, SIGCSE '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 1163–1169. URL: https://doi.org/10.1145/3408877.3432539. doi:10.1145/3408877.3432539.

[17] P. Denny, D. H. Smith IV, M. Fowler, J. Prather, B. A. Becker, J. Leinonen, Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills, arXiv preprint arXiv:2403.06050 (2024).

[18] S. Haller, A. Aldea, C. Seifert, N. Strisciuglio, Survey on automated short answer grading with deep learning: from word embeddings to transformers, arXiv preprint arXiv:2204.03503 (2022).

[19] J. Lin, Z. Han, D. R. Thomas, A. Gurung, S. Gupta, V. Aleven, K. R. Koedinger, How can i get it right? using gpt to rephrase incorrect trainee responses, arXiv preprint arXiv:2405.00970 (2024).

[20] A. B. L. Narayanan, P. Oli, J. Chapagain, M. Hassany, R. Banjade, P. Brusilovsky, V. Rus, Explaining code examples in introductory programming courses: LLM vs humans, in: AI for Education: Bridging Innovation and Responsibility at the 38th AAAI Annual Conference on AI, 2024. URL: https://openreview.net/forum?id=zImjfZG3mw.

[21] S. Banerjee, A. Lavie, Meteor: An automatic metric for mt evaluation with improved correlation with human judgments, in: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, 2005, pp. 65–72.

[22] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, Y. Artzi,

Bertscore: Evaluating text generation with bert, arXiv preprint arXiv:1904.09675 (2019).

[23] M. Popović, chrf: character n-gram f-score for automatic mt evaluation, in: Proceedings of the tenth workshop on statistical machine translation, 2015, pp. 392–395.

[24] R. Hosseini, K. Akhuseyinoglu, P. Brusilovsky, L. Malmi, K. Pollari-Malmi, C. Schunn, T. Sirkiä, Improving engagement in program construction examples for learning python programming, International Journal of Artificial Intelligence in Education 30 (2020) 299–336. URL: https://doi.org/10.1007/s40593-020-00197-0. doi:10.1007/s40593-020-00197-0.

[25] R. Hosseini, P. Brusilovsky, Javaparser: A fine-grain concept indexing tool for java problems, in: CEUR Workshop Proceedings, volume 1009, University of Pittsburgh, 2013, pp. 60–63.