Automated Assessment of Student Self-Explanation in Code Comprehension Using Pre-Trained Language Models

Jeevan Chapagain and Vasile Rus

University of Memphis
Memphis, TN, USA, 38152
jchpgain@memphis.edu, vrus@memphis.edu

Abstract

Assessing students' responses, especially natural language responses, is a major challenge in education. In general, in education contexts, automatically evaluating what learners do or say is important as it enables personalized instruction, e.g., based on what the learner knows tailored tasks and feedback are given to the learner. Recently, deep learning techniques led to state-of-the-art methods in NLP such as transformer-based methods which resulted in significant performance improvements for many NLP tasks such as text classification and question answering. However, there is not much work exploring such methods for assessing students' free answers, particularly in the context of code comprehension, which brings additional challenges as the student explanations include code references as well. This paper explores the potential of applying automated assessments methods using transformers to code comprehension. We fine-tuned pre-trained transformer models, including BERT, RoBERTa, CodeBERT, and SciBERT, to see how well they can automatically judge students' responses to code comprehension tasks. Our results demonstrate that these models can significantly enhance the accuracy and reliability of automated assessments, offering insights into how the latest NLP techniques can be leveraged in computer science education to support personalized learning experiences.

Introduction

When students learn, monitoring their knowledge state and other states such as their emotional state, i.e., inferring a learner model, is crucial as it enables tailored instruction to individual needs which is needed in order to induce optimal student learning gains. In adaptive instructional systems such as Intelligent Tutoring Systems (ITS) (Rus et al. 2013), the learner model needs to be inferred automatically and implies assessing the learner's mastery level or knowledge state with respect to the target domain as well as other states, as noted. Assessing students' responses during instructional tasks helps detect learners' knowledge states with particular key concepts covered by a particular instructional tasks and also provides insights into students' overall mastery level of the domain (Harlen et al. 1992). The learner model is critical for both macro-adaptivity, i.e., selecting learning tasks appropriate for the learner's mastery level, e.g., following a Zone of Proximal Development (ZPD) approach, and micro-adaptivity which entails providing targeted support within a specific task such as when the learner is engaged in a problem-solving or code comprehension task.

Our work presented here has been conducted within the context of source code comprehension ITSs, or shortly code comprehension ITSs. Reading code and understanding what it does, i.e. code comprehension, is a critical skill for both learners as well as professionals. For learners, it is crucial to acquire the ability to read and understand code, as learning computer programming involves exploring through reading many code examples provided by the instructor or available in intro-to-programming textbooks or other sources (Cunningham et al. 2022; Wang et al. 2020). Similarly, reading code to understand what it does is a major skill and activity for professional programmers. Professional developers spend approximately 70% of their time understanding code during software maintenance (Rugaber 2000). As O'Brien (O'brien 2003) notes, comprehending source code is vital for tasks such as maintenance, reuse, migration, reengineering, or enhancement. Thus, enhancing students' code comprehension skills would significantly benefit their academic and professional careers.

Research on code comprehension strategies and on developing ITSs for code comprehension have focused on prompting for self-explanation as an instructional strategy. Students are asked to read code examples and then articulate in their own words their understanding of the code. Self-explanation is generally an effective learning strategy. Self-explanation theories and empirical evidence (Chi et al. 1989; Chi 2000) suggest that students who self-explain the learning material achieve deeper understanding and greater learning gains. In the process of self-explanation, students articulate their thought processes during educational activities (Chi et al. 1989; Chi 2000). When coupled with strategies based on socio-constructivist theories of learning, self-explanation can be an effective strategy because it involves learners interactively verbalizing their comprehension as they engage in learning tasks with a significantly more knowledgeable other, e.g., a tutor, and receive support in the form of feedback, e.g., positive feedback such as *Great job!* and hints as needed resulting in a scaffolded self-explanation instructional strategy. Following constructivist theories of learning the learner is allowed to construct

their own knowledge by themselves as much as possible and only offered help in the form of hints when floundering (Aleven and Koedinger 2002; VanLehn, Jones, and Chi 1992). Moreover, self-explanations hold personal relevance for the learner, being self-initiated and generated, thereby enriching the learning experience's significance (Roy and Chi 2005). The positive effects of self-explanations have been shown in different domains like physics (Conati and VanLehn 2000), and programming (Rus et al. 2021).

Based on this rich evidence of the effectiveness of scaffolded self-explanation, code comprehension tutors (Rus et al. 2019) have recently been developed that interactively scaffold learners' code comprehension processes by eliciting self-explanations and providing feedback. Automatically assessing students' self-explanations of code is a critical component in such tutors. A widely adopted approach relies on semantic similarity methods where students' self-explanations are compared to ideal self-explanations provided by the experts. Although significant research has been conducted in automated assessment for short answers (Banjade et al. 2016) and essays (Mujeeb, Pardeshi, and Ghongane 2010), less work has been done on assessing free self-explanations during source code comprehension activities, which is our research focus.

It is important to note that code explanations present unique challenges compared to other free-text scoring tasks. Unlike general free-text responses, code explanations incorporate domain-specific language and require a deep understanding of both natural language and programming language concepts. They must be closely aligned with the structure and logic of the corresponding code example. Furthermore, code explanations need to be assessed not only for content but also for how well they capture the precision and conciseness required in programming. These unique aspects require specialized approaches for automated assessment, which this study aims to address.

This paper seeks to bridge the gap in automated assessment for source code comprehension by exploring the application of transformer-based models through fine-tuning them to evaluate students' programming self-explanations. Specifically, we focus on line-by-line self-explanation generated by students for JAVA code examples, leveraging pre-trained models, namely, BERT, RoBERTa, DistilBERT, CodeBERT, and SciBERT. These models have shown promise in various NLP tasks (Sanh et al. 2019; Feng et al. 2020; Beltagy, Lo, and Cohan 2019). The rationale behind selecting these specific transformer models is detailed in the methodology section.

This study contributes to the fields of computer science education and ITS development by addressing the lack of research on assessing self-explanations during source code comprehension. We expect that our findings will not only enhance our understanding of how students comprehend code but also inform the design of more effective educational tools that support personalized learning experiences.

Related Work

The automated assessment of students' natural language responses has evolved significantly, with methodologies transitioning from hand-crafted features to advanced neural network approaches. In this section, we review some of the work that has been done in the automated assessment of students' responses.

Earlier approaches utilized by researchers to automatically assess students' responses included hand-crafted features like lexical similarity (Dzikovska, Nielsen, and Brew 2012), vector-based similarity (Sultan, Salazar, and Sumner 2016), and n-gram features (Heilman and Madnani 2013). These approaches laid the groundwork for understanding textual responses but lacked the depth required for assessing more challenging responses in the context of more complex tasks such as code comprehension.

With the introduction of neural networks, various deep learning-based approaches were introduced to automate the assessment of freely generated, natural language responses (Riordan et al. 2017; Conneau et al. 2017). Kumar et al. (Kumar, Chakrabarti, and Roy 2017) introduced Siamese bidirectional LSTMs with an Earth-Mover distance pooling layer for Automated Short Answer Grading (ASAG), demonstrating the potential of deep learning in capturing nuanced semantic relationships. Similarly, Prabhudesai et al. (Prabhudesai and Duong 2019) used a Siamese bidirectional LSTM neural network-based regression, combining deep learning with feature engineering to improve assessment accuracy. These studies highlight the neural networks' ability to understand complex textual data, suggesting a foundation upon which assessment of self-explanation in code comprehension could be built.

Sung et al.(Sung et al. 2019) pre-trained BERT with domain-specific data to improve its effectiveness, representing the latest advancement in NLP. This method's success in various tasks, including ASAG, showed the potential of transformer models to adapt to specialized domains. Liu et al. (Liu et al. 2019a) and Camus et al. (Camus and Filighera 2020) further explored transformer capabilities, employing attention mechanisms and fine-tuning strategies to understand the nuanced semantics of student responses, supporting the idea that these models might more accurately represent the semantic relationship. Nadeem et al. (Nadeem et al. 2019) developed an LSTM-based scoring architecture for essays that leveraged pre-computed contextual embeddings from a static BERT model as initialization parameters.

Candor (Condor 2020) specifically adapted BERT for ASAG, using Cohen's Kappa to compare the agreement between automated systems and human evaluators. Their findings revealed that pre-trained models like BERT yield more consistent ratings, similar to those of human evaluators, highlighting the potential for these models to provide reliable assessments in new domains, such as code comprehension. Lun et al. (Lun et al. 2020) employed a multiple-data augmentation strategy in combination with a fine-tuned BERT model to provide automated short-answer scoring.

A study by Khayi et al. (Khayi, Rus, and Tamang 2021) in 2021 investigated the use of fine-tuned pre-trained transformer models to evaluate open-ended student answers in physics. They integrated this approach into an interactive, dialogue-based ITS. Both of their work showed that fine-tuning pre-trained language models for short answer grad-

ing lead to remarkable performance improvement compared to traditional approaches. This technique leverages the power of large-scale pre-trained models and adapts them to the specific task of evaluating student responses. Ghavidel et al. (Ghavidel, Zouaq, and Desmarais 2020) developed an innovative assessment system that merges BERT with XLNET (Extra Long-Network) using autoregressive pre-training. Unlike traditional approaches, their model functions without manually engineered features and eliminates the need for question-based training or inputs.

Despite these advances, the direct application of these models to assessing self-explanations in source code comprehension remains largely unexplored. Although Chapagain (Chapagain et al. 2022) and Fowler (Fowler et al. 2021) have explored automated assessment of code explanations, their reliance on traditional machine learning approaches such as textual features, bag-of-word, and bigram models have limited capabilities compared to transformer-based models. Traditional machine learning methods often rely on predefined features and patterns which can limit their ability to fully capture the complexities of programming logic or semantics in student responses.

This gap presents a significant opportunity to explore promising transformer-based models for assessing student response in code comprehension, aiming to advance the field in the automated assessment of free student responses.

Data Collection

We used a publicly available dataset called SelfCode 2.0(Lekshmi-Narayanan et al. 2024) which was collected using an online learning tool named PCEX (Hicks et al. 2020) which offers interactive worked examples and enables students to learn programming by interactively exploring code examples line by line. The original PCEX platform was modified to encourage students to provide explanations for each line of code, aiming to obtain authentic self-explanations from them. Specifically, they were presented with 10 different JAVA examples and asked to read and explain each line, resulting in a total of 3,019 self-explanations. The explanations for the experts were provided by final year Computer Science Ph.D. students who were also asked to explain each line of code. Table 2 illustrate expert and corresponding student explanations with their semantic similarity ratings.

Explanations	Word Count (μ)	Word Count (σ)
Student	16.41	9.96
Expert	24.78	19.56

Table 1: Statistics of student explanation and expert explanation

Two Computer Science Ph.D. students annotated the student and expert explanations regarding semantic similarity. They assessed how similar the students' self-explanations are to the corresponding expert explanations on a scale of 1-5, with 1 being not similar and 5 being the most similar. This annotation process was designed to accommodate various explanation pairs, leading to four distinct cases based on the explanations provided by both students and experts. Each case represents a different scenario of sentence pairings and was rated accordingly to capture the similarity between the student and expert explanations as presented below. Examples of each 4 cases can be seen in Table 2.

- Case 1: When both the student and the expert provide a one-sentence explanation each, the sentence pair was assigned a singular rating reflecting their similarity. For example, a rating of [(4)] denotes a single sentence pair between the expert and the student.
- Case 2: If the student provides a single sentence as an explanation whereas the expert provided multiple sentences for the same line of code, the rating was expressed as [(3,1)], with each score representing the comparison of the student sentence to each of the expert sentences, highlighting the variance in similarity across the multiple expert sentences.
- Case 3: When the student provides multiple sentences against a single expert sentence, each student sentence was rated for its similarity to the expert's sentence, leading to individual ratings such as [(4), (1)], indicating how each student sentence compares to the single expert sentence.
- Case 4: For pairs where both provide multiple sentences, each student's sentence was compared with each expert sentence, resulting in pairs of scores (e.g., [(2,2), (1,1)]). This method details the similarity between all pairs of student sentences and the expert sentences.

To create a single score for each annotator, we first used a maximum-of-maxima (max-of-max) approach, selecting the highest score from all comparisons to highlight the most significant match between the student and expert explanations. This leads to an overly optimistic score. Our second criterion was the maximum of averages (max of averages), which, as the name implies, computes the average similarity score for each group or set of sentence pairs and then selects the maximum among these averages. This can be interpreted as identifying the group whose average similarity score is highest compared to others. That is, this method selects as the overall, average score of the sentence which is most similar on average with all sentences in the expert explanations. This can also be interpreted as the sentence that is most consistently similar with all the sentences in the expert explanation on average. This also leads to an optimistic assessment of students' explanations. It should be noted that these optimistic approaches aim to mitigate what we consider the biggest risk: not giving students credit when they actually deserve it. That is, given that even state-of-the-art NLP techniques are not perfect, an automated method can indicate a student is incorrect when, in fact, they are correct (false negative). The optimistic approaches minimize this worst-case scenario, and the price to pay is that, at times, such approaches give students a more generous score than they deserve. We believe this trade-off is appropriate given the current state of the technology.

The third aggregating method uses an average of maxima (avg of max). It first identifies the maximum similarity score

Line of Code	Student Explanation	Expert Explanation	Similarity Score
Point point = new Point();	Creates a class point and creates a variable within that class called point and sets it equal to a method point.	We need to create a Point object to represent a point in the Euclidean plane	[(4)]
private int y;	This declares a private int variable called y	The instance variables are declared as private to prevent direct access to them from outside the class. In this way, no unexpected modifications to a Point object's data are possible.	[(3,1)]
int[] values = {5, 8, 4, 78, 95, 12, 1, 0, 6, 35, 46};	This line creates the integer array with the values. you need this to achieve the goal bc you need an array to look in	We declare an array of values to hold the numbers.	[(4), (1)]
if (num == previous) {	If loop that compares num to previous. If they are equal, then it will print the statement.	We check whether the number that the user entered is a duplicate of the previous number that the user entered. To determine if the two numbers are duplicates, we need to test whether they are equal.	[(2,2),(1,1)]

Table 2: Comparison of Student and Expert Explanations with Similarity Scores

within each group or sentence pairs. Then, it calculates the average of these maximum scores across all groups or sentences in the student explanation. Although this approach is also optimistic, it is moderately so, as it averages maximum scores for each student sentence with respect to each expert sentence. If some student sentences have low similarity with expert sentences, they will bring down the overall average of maxima. This method balances capturing the strongest similarities while also acknowledging weaker points in the student's explanation. It highlights the best matching parts without completely overlooking less aligned sections.

Table 1 shows the descriptive statistics of expert and student explanations. To ensure consistency with semantic similarity scores, which range from 0 to 1, we have adjusted the single-rated scores accordingly. A human rating of 1 is normalized to 0, a rating of 2 becomes 0.25, and so forth, culminating in a rating of 5 is normalized to 1.

Methodology

To choose what pre-trained transformer models to use, we considered various aspects such as training data, model architecture, and pre-training objectives. Our goal was to explore which models are better suited for our specific task of automated assessment of students' self-explanations during code comprehension tasks.

BERT: BERT (Devlin et al. 2018) is designed for deep bidirectional learning from unlabeled text which is trained on a corpus of news articles and Wikipedia. For our regression tasks, we fine-tuned BERT by adding a regression head on top of the pre-trained model. This process involves training the entire model, including the newly added regression layer, on our specific dataset of student and expert explanations. This process involves embedding the [CLS] token from BERT's final output and passing it through a fully connected layer that predicts a continuous value. This modification enables BERT to effectively handle tasks involving numerical predictions, leveraging its strong language-understanding capabilities.

RoBERTa : RoBERTa (Liu et al. 2019b) shares the same architecture as BERT but differs in its training approach. It incorporates modifications such as dynamic masking and is trained on a larger dataset of English language texts, including books, news articles, and web content. Furthermore, RoBERTa employs a different encoding mechanism compared to BERT using byte-level Byte-Pair Encoding (BPE) instead of WordPiece tokenization. This allows RoBERTa to handle a larger vocabulary and potentially capture more nuanced relationships between words and subwords.

DistilBERT: DistilBERT (Sanh et al. 2019) is a streamlined version of BERT, offering similar performance but with fewer parameters, making it faster and more efficient. It is trained on the same dataset as BERT. To adapt DistilBERT for regression tasks, we fine-tuned DistilBERT using the same approach as BERT, replacing the classification layer with a regression head.

CodeBERT: CodeBERT (Feng et al. 2020) is a variant of the BERT model, specifically designed to understand both natural language and programming code. It is pre-trained on pairs of natural languages and code in six programming languages: Python, Java, JavaScript, PHP, Ruby, and Go.

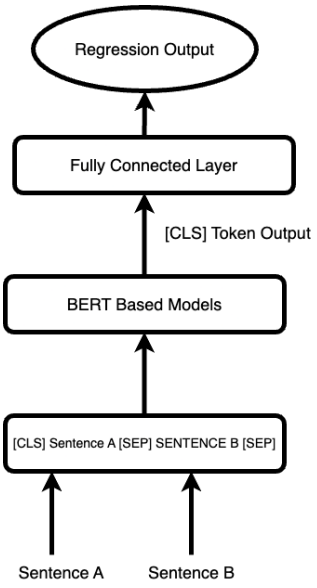

Figure 1: BERT based model with regression head for semantic similarity prediction

SciBERT: SciBERT (Beltagy, Lo, and Cohan 2019) is a specialized version of the BERT model, tailored for scientific text. It is pre-trained on a large corpus of scientific literature from the fields of computer science and biomedicine. This allows SciBERT to effectively capture domain-specific concepts.

For the fine-tuning process, we input the students' and expert's explanations into the various transformer-based models. The output of the fine-tuned models is a semantic similarity score between 0 and 1, indicating how similar the student's sentence is to an expert's sentence. This score is essentially a regression value predicted by the model after fine-tuning. The classification layer was replaced with a regression head to adapt the pre-trained models for this regression task. Specifically, the [CLS] token output from the final transformer layer was passed through a fully connected layer to predict the continuous similarity score. During fine-tuning, the weights of both the pre-trained BERT-based model layers and the new regression layer are updated through backpropagation, allowing the model to adapt to our specific task while retaining its pre-trained knowledge. Figure 1 shows the architecture of the BERT based transformer model with the regression head.

Experiments and Results

In our experiment, we fine-tuned transformed models for 10 epochs with a batch size of 16, a learning rate of 0.00002, and AdamW as an optimizer. This approach was chosen to optimize learning efficiently while preventing overfitting. Our experiments were carried out on Google Colab, utilizing its T4 GPU. We used Pearson and Spearman correlation coefficients as evaluation metrics to analyze how effectively the models' assessments aligned with human judgments. We applied an 80-10-10 split procedure using random selection to split the data sets into training (1812 instances), test (604 instances), and validation (604 instances).

Models	Max_of_Max		Avg_of_Max		Max_of_Avg	
	P	S	P	S	P	S
TF-IDF	.310	.290	.313	.293	.313	.293
BERTScore	.377	.346	.378	.347	.397	.385
USE	.307	.275	.306	.204	.306	.274
BERT	.677	.631	.772	.765	.703	.662
RoBERTa	.647	.615	.726	.728	.645	.618
DistilBERT	.716	.665	.767	.759	.721	.670
CodeBERT	.656	.626	.734	.739	**.738**	**.739**
SciBERT	**.733**	**.681**	**.784**	**.775**	.736	.686

P: Pearson, S: Spearman

Table 3: Fine-tuned models performance along with baseline for different aggregation

For our experiment, we used three baseline methods: TF-IDF, BERTScore (Zhang et al. 2019), and Universal Sentence Encoder(USE) (Cer et al. 2018) to calculate the semantic similarity scores between expert responses and student responses. TF-IDF represents responses as term frequency-inverse document frequency vectors, with similarities computed using cosine distance between these vectors. BERTScore leverages BERT's contextual embeddings to compute token-wise cosine similarities between responses, which are then aggregated into a final score. Similarly, USE converts sentences into fixed-length vectors and employs cosine similarity for comparison. These baselines are crucial as they give us a standard to measure the improvements made by our fine-tuned transformer models.

Table 3 shows the results of fine-tuning pre-trained transformer models compared to a baseline model when explanations are passed without splitting. We used a holistic approach to evaluate entire explanations at once because it allows us to understand how all parts of an explanation work together, mimicking the natural way humans assess explanations, considering not just the individual elements but how they cohesively form a complete argument.

The result shows that all pre-trained models outperform the baseline. SciBERT has the best performance across all metrics evaluated in both the max_of_max and avg_of_max aggregation methods, achieving Pearson's correlation coefficients of 0.733 and 0.784, respectively, and Spearman's correlation coefficients of 0.681 and 0.775, respectively. These high correlation scores indicate that SciBERT's assessments closely match human evaluations, an essential factor in accurately evaluating student responses.

Our analysis reveals interesting performance variations between SciBERT and CodeBERT in assessing student explanations. While SciBERT achieved higher scores in max-of-max (0.733 vs 0.656) and avg-of-max (0.784 vs 0.734) metrics, CodeBERT performed marginally better in max-of-avg assessment (0.739 vs 0.736). SciBERT's superior performance can be attributed to its training in scientific literature, which provides better handling of complex technical

explanations and terminology variations found in student responses. Its exposure to diverse scientific writing styles enables better bridging between expert explanations and alternative student phrasings.

In contrast, CodeBERT's training on code-natural language pairs, while effective for direct code-language relationships, may not fully capture the range of explanatory patterns in student responses. CodeBERT's stronger performance in max-of-avg assessment suggests better capability in identifying partial matches and technically accurate but incomplete explanations. These complementary strengths suggest potential benefits in combining both models through ensemble methods or specialized fine-tuning approaches for future developments.

All transformer models performed significantly compared to the baseline, but their effectiveness varies depending on the aggregation method used. For instance, under the avg_of_max aggregation, models like RoBERTa and DistilBERT also excelled, with DistilBERT reaching a Pearson correlation of 0.767 and RoBERTa achieving approximately 0.726. This indicates that the avg_of_max method may better reflect the models' detailed understanding compared to the max_of_avg and max_of_max methods, where the performance generally appears to be lower. The difference in how well each model performs based on the aggregation method used points to the advantage of using a variety of models for different kinds of assessment tasks in adaptive learning systems. By choosing the right model for the task, we can make the most of what each model does best, leading to more precise and fair evaluations of student learning.

We also performed an error analysis to analyze the performance of the model. The models generally performed well on student explanations that closely matched the expert's explanations. However, it struggled with student explanations that used alternative but correct phrasing to describe the code. Additionally, the models had difficulty accurately assessing longer, multi-sentence explanations, particularly when the student elaborated on concepts not explicitly mentioned in the expert explanation. This suggests that the models may need further refinement to fully capture the nuances while explaining code examples. These findings of our error analysis highlight the need for future work to focus on improving the models' ability to better handle the diversity of valid explanations.

In summary, fine-tuning pre-trained transformer models has shown good results in automatically assessing student responses to code comprehension. SciBERT showed better performance in the case of both the max_of_max and avg_of_max methods, closely matching human evaluation. CodeBERT is also effective, especially in the max_of_avg method, showing that it is good at assessing. The performance of these models along with other models such as BERT, RoBERTa, and DistillBERT highlights the need to choose the right model and method of aggregation for the most accurate student assessments in education. These findings can help adaptive learning systems use the best NLP tools for better and more personalized assessments in computer science education.

Conclusion

This study demonstrates the potential of using fine-tuned transformer models such as SciBERT and CodeBERT to automatically assess students' code explanations. Our analysis reveals that the Average-of-Max aggregation method consistently yields the strongest performance across different models, with SciBERT achieving the highest scores (Pearson: .784, Spearman: .775). This aggregation strategy appears to strike an optimal balance in assessment, neither overly generous nor unnecessarily strict in evaluating student responses against reference explanations. Such balanced evaluation is crucial in educational settings where we want to fairly acknowledge students' understanding while maintaining high standards.

Integrating these models with the Average-of-Max aggregation method into intelligent tutoring systems (ITSs) could enable more accurate and personalized feedback on students' code comprehension abilities. Instructors could leverage these assessments to identify knowledge gaps and provide targeted instruction tailored to individual student needs. However, a key limitation is the reliance on a single expert reference explanation, which may not account for the natural variance in valid student responses. Incorporating multiple expert references could further enhance the effectiveness of our chosen aggregation strategy.

Our current work doesn't take account of context such as providing the goal of the program. Incorporating program goals and other contextual information could significantly enhance the model's ability to evaluate student's responses more accurately which we plan to explore in the future. Additionally, while the models performed well on this specific dataset, their generalizability to other programming languages or domains needs further investigation. In addition, we also want to look at the interpretability of these models, as it would help ITS to provide more personalized feedback to learners, making learning more effective. Moreover, ethical considerations around fairness, transparency, and accountability must also be addressed before adopting these technologies and models into educational settings. Nonetheless, this work paves the way for enhancing adaptive learning experiences by automating the evaluation of students' code explanations.

Acknowledgments

This work has been supported by the following grants awarded to Dr. Vasile Rus: the Learner Data Institute (NSF award 1934745); CSEdPad (NSF award 1822816); and iCODE (IES award R305A220385). The opinions, findings, and results are solely those of the authors and do not reflect those of NSF or IES.

References

Aleven, V. A.; and Koedinger, K. R. 2002. An effective metacognitive strategy: Learning by doing and explaining with a computer-based cognitive tutor. *Cognitive science*, 26(2): 147–179.

Banjade, R.; Maharjan, N.; Niraula, N. B.; Gautam, D.; Samei, B.; and Rus, V. 2016. Evaluation dataset (DT-Grade)

and word weighting approach towards constructed short answers assessment in tutorial dialogue context. In *Proceedings of the 11th workshop on innovative use of nlp for building educational applications*, 182–187.

Beltagy, I.; Lo, K.; and Cohan, A. 2019. SciBERT: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*.

Camus, L.; and Filighera, A. 2020. Investigating transformers for automatic short answer grading. In *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part II 21*, 43–48. Springer.

Cer, D.; Yang, Y.; Kong, S.-y.; Hua, N.; Limtiaco, N.; John, R. S.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Chapagain, J.; Tamang, L.; Banjade, R.; Oli, P.; and Rus, V. 2022. Automated Assessment of Student Self-explanation During Source Code Comprehension. In *The International FLAIRS Conference Proceedings*, volume 35.

Chi, M. T. 2000. Self-explaining expository texts: The dual processes of generating inferences and repairing mental models. *Advances in instructional psychology*, 5: 161–238.

Chi, M. T.; Bassok, M.; Lewis, M. W.; Reimann, P.; and Glaser, R. 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science*, 13(2): 145–182.

Conati, C.; and VanLehn, K. 2000. Further results from the evaluation of an intelligent computer tutor to coach self-explanation. In *International Conference on Intelligent Tutoring Systems*, 304–313. Springer.

Condor, A. 2020. Exploring automatic short answer grading as a tool to assist in human rating. In *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part II 21*, 74–79. Springer.

Conneau, A.; Kiela, D.; Schwenk, H.; Barrault, L.; and Bordes, A. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Cunningham, K.; Qiao, Y.; Feng, A.; and O'Rourke, E. 2022. Bringing" High-Level" Down to Earth: Gaining Clarity in Conversational Programmer Learning Goals. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, 551–557.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dzikovska, M. O.; Nielsen, R. D.; and Brew, C. 2012. Towards effective tutorial feedback for explanation questions: A dataset and baselines. In *Proceedings of the 2012 conference of the North American Chapter of the Association for Computational Linguistics: Human language technologies*, 200–210. Association for Computational Linguistics.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Fowler, M.; Chen, B.; Azad, S.; West, M.; and Zilles, C. 2021. Autograding" Explain in Plain English" questions using NLP. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 1163–1169.

Ghavidel, H. A.; Zouaq, A.; and Desmarais, M. C. 2020. Using BERT and XLNET for the Automatic Short Answer Grading Task. In *CSEDU (1)*, 58–67.

Harlen, W.; Gipps, C.; Broadfoot, P.; and Nuttall, D. 1992. Assessment and the improvement of education. *The curriculum journal*, 3(3): 215–230.

Heilman, M.; and Madnani, N. 2013. ETS: Domain adaptation and stacking for short answer scoring. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, 275–279.

Hicks, A.; Akhuseyinoglu, K.; Shaffer, C.; and Brusilovsky, P. 2020. Live catalog of smart learning objects for computer science education. In *Sixth SPLICE Workshop*.

Khayi, N. A.; Rus, V.; and Tamang, L. 2021. Towards improving open student answer assessment using pretrained transformers. In *The International FLAIRS Conference Proceedings*, volume 34.

Kumar, S.; Chakrabarti, S.; and Roy, S. 2017. Earth Mover's Distance Pooling over Siamese LSTMs for Automatic Short Answer Grading. In *IJCAI*, 2046–2052.

Lekshmi-Narayanan, A.-B.; Chapagain, J.; Brusilovsky, P.; and Rus, V. 2024. SelfCode 2.0: Annotated Corpus of Student Self- Explanations to Introductory JAVA Programs in Computer Science.

Liu, T.; Ding, W.; Wang, Z.; Tang, J.; Huang, G. Y.; and Liu, Z. 2019a. Automatic short answer grading via multiway attention networks. In *Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part II 20*, 169–173. Springer.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019b. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lun, J.; Zhu, J.; Tang, Y.; and Yang, M. 2020. Multiple data augmentation strategies for improving performance on automatic short answer scoring. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 13389–13396.

Mujeeb, A.; Pardeshi, M.; and Ghongane, B. 2010. Comparative assessment of multiple choice questions versus short essay questions in pharmacology examinations. *Indian journal of medical sciences*, 64(3): 118.

Nadeem, F.; Nguyen, H.; Liu, Y.; and Ostendorf, M. 2019. Automated essay scoring with discourse-aware neural models. In *Proceedings of the fourteenth workshop on innovative use of NLP for building educational applications*, 484–493.

O'brien, M. P. 2003. Software comprehension–a review & research direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report*.

Prabhudesai, A.; and Duong, T. N. 2019. Automatic short answer grading using siamese bidirectional LSTM based regression. In *2019 IEEE international conference on engineering, technology and education (TALE)*, 1–6. IEEE.

Riordan, B.; Horbach, A.; Cahill, A.; Zesch, T.; and Lee, C. 2017. Investigating neural architectures for short answer scoring. In *Proceedings of the 12th workshop on innovative use of NLP for building educational applications*, 159–168.

Roy, M.; and Chi, M. T. 2005. The self-explanation principle in multimedia learning. *The Cambridge handbook of multimedia learning*, 271–286.

Rugaber, S. 2000. The use of domain knowledge in program understanding. *Annals of Software Engineering*, 9(1): 143–192.

Rus, V.; Akhuseyinoglu, K.; Chapagain, J.; Tamang, L.; and Brusilovsky, P. 2021. Prompting for Free Self-Explanations Promotes Better Code Comprehension.

Rus, V.; Brusilovsky, P.; Fleming, S.; Tamang, L.; Akhuseyinoglu, K.; Barria-Pineda, J.; Ait-Khayi, N.; and Alshaikh, Z. 2019. An Intelligent Tutoring System for Source Code Comprehension. In *The 20th International Conference on Artificial Intelligence in Education, June 25-29, Chicago, IL, USA*.

Rus, V.; D'Mello, S.; Hu, X.; and Graesser, A. 2013. Recent advances in conversational intelligent tutoring systems. *AI magazine*, 34(3): 42–54.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sultan, M. A.; Salazar, C.; and Sumner, T. 2016. Fast and easy short answer grading with high accuracy. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1070–1075.

Sung, C.; Dhamecha, T.; Saha, S.; Ma, T.; Reddy, V.; and Arora, R. 2019. Pre-training BERT on domain resources for short answer grading. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 6071–6075.

VanLehn, K.; Jones, R. M.; and Chi, M. T. 1992. A model of the self-explanation effect. *The journal of the learning sciences*, 2(1): 1–59.

Wang, W.; Rao, Y.; Zhi, R.; Marwan, S.; Gao, G.; and Price, T. W. 2020. Step tutor: Supporting students through step-by-step example-based feedback. In *Proceedings of the 2020 ACM conference on innovation and technology in computer science education*, 391–397.

Zhang, T.; Kishore, V.; Wu, F.; Weinberger, K. Q.; and Artzi, Y. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.