Automated Assessment of Student Self-explanation During Source Code Comprehension

Jeevan Chapagain, Lasang Tamang, Rabin Banjade, Priti Oli, Vasile Rus
Department of Computer Science, Institute of Intelligent System
University of Memphis, Memphis, TN, USA
{jchpgain, ljtamang, rbnjade1, poli,vrus}@memphis.edu

Abstract

This paper presents a novel method to automatically assess self-explanations generated by students during code comprehension activities. The self-explanations are produced in the context of an online learning environment that asks students to freely explain Java code examples line-by-line. We explored a number of models consisting of textual features in conjunction with machine learning algorithms such as Support Vector Regression (SVR), Decision Trees (DT), and Random Forests (RF). Support Vector Regression (SVR) performed best having a correlation score with human judgments of 0.7088. The best model used a combination of features such as semantic measures obtained using a Sentence BERT pre-trained model and from previously developed semantic algorithms used in a state-of-the-art intelligent tutoring system.

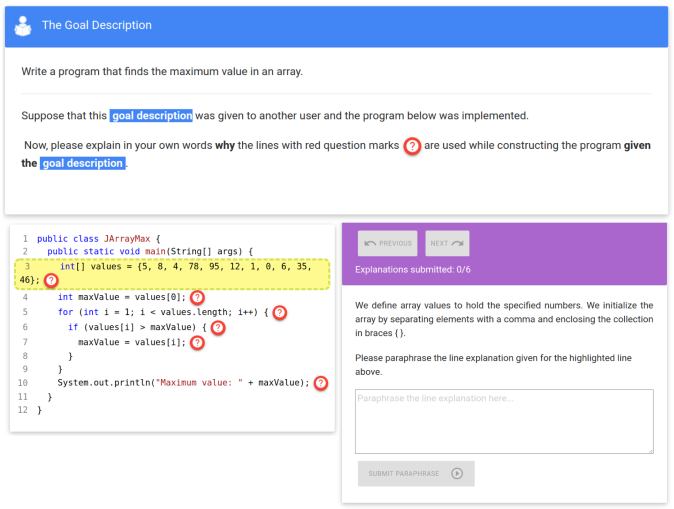
Introduction

Source code comprehension refers to the process of understanding a code example, ”a process in which an individual constructs his or her mental representation of the program” (Schulte et al. 2010). Code comprehension is essential for both professionals and beginners, e.g., students who learn programming. Indeed, students learning computer programming spend a significant portion of their time reading or reviewing someone else’s code (e.g., source code examples from a textbook or provided by the instructor). Software professionals spend at least half of their time analyzing software artifacts in an attempt to comprehend computer source code. Reading code is the most time-consuming activity during software maintenance, consuming 70% of the total life-cycle cost of a software product (Rugaber 2000; Buse and Weimer 2008). O’Brien (O’brien 2003) notes that source code comprehension is required when a programmer maintains, reuses, migrates, re-engineers, or enhances a software system. Thus, helping students strengthen their source code comprehension skills will have a significant impact on their academic as well as a professional career.

One instructional strategy to improve code comprehension and learning of programming is to prompt students to freely self-explain the code they are trying to comprehend. Self-explanation theories (Chi 2000) indicates that students who engage in self-explanations while learning are better learners. The positive impact of self-explanation has previously been demonstrated in different domains like physics (Conati and VanLehn 2000), math (Aleven and Koedinger 2002), and programming (Bielaczyc, Pirolli, and Brown 1995; Tamang et al. 2020; Rus et al. 2021). This work is part of a larger project meant to develop an advanced education technology that scaffolds learners’ code comprehension processes by eliciting self-explanations and providing feedback, e.g., positive feedback such as *Great!* followed by an assertion reinforcing students’ understanding or, if the case, negative feedback followed by, for instance, correcting a misconception. Figure 1 shows the prompts given to students and examples of the corresponding lines of code. A key component of this technology is automatically assessing students’ self-explanations which we address using a semantic similarity approach in which students’ self-explanations are compared to ideal self-explanations provided by experts.

Although many studies have been conducted to measure the semantic similarity between two texts, not much work has been done in the field of computer programming, and in particular, in the context of source code comprehension activities. The correctness of students’ responses must be assessed in order to provide adequate feedback. The goal of the work presented here is to investigate a number of methods that combine machine learning and natural language techniques to automatically assess student generated self-explanations of code examples. As noted, the approach we take is to measure how semantically similar the student self-explanations are to benchmark explanations provided by experts.

More specifically, we present a set of novel methods to compute the semantic similarity between sentence level self-explanations of JAVA code examples when students are asked to freely self-explain the code line-by-line. The general approach consists of a feature engineering phase to extract textual features and then, in a second phase, use those features in a number of machine learning models to compute a normalized semantic similarity score between sentence level self-explanations and expert-generated explanations of the same lines of code.



Write a program that finds the maximum value in an array.

Suppose that this **goal description** was given to another user and the program below was implemented.

Now, please explain in your own words **why** the lines with red question marks ? are used while constructing the program **given the goal description**.

```java
public class JArrayMax {
  public static void main(String[] args) {
    int[] values = {5, 8, 4, 78, 95, 12, 1, 0, 6, 35, 46};
    int maxValue = values[0];
    for (int i = 1; i < values.length; i++) {
      if (values[i] > maxValue) {
        maxValue = values[i];
      }
    }
    System.out.println("Maximum value: " + maxValue);
  }
}
```

Explanations submitted: 0/6

We define array values to hold the specified numbers. We initialize the array by separating elements with a comma and enclosing the collection in braces { }.

Please paraphrase the line explanation given for the highlighted line above.

Figure 1: PCEX web application for collecting line-by-line student self explanation

Semantic Similarity as an Approach to Natural Language Understanding

Measuring the semantic similarity of texts can be framed as quantifying the degree of semantic similarity between a given pair of texts, such as two words or two sentences or even larger texts (Agirre et al. 2015; Rus et al. 2008). Similarity scores typically range between 0 to 1 (normalized scores), 0 meaning no similarity at all, whereas 1 meaning most similar. Semantic similarity is widely used in the field of Natural Language Processing (NLP) for various tasks like text summarization, information retrieval, sentiment analysis and so on. Semantic similarity between two texts can also be framed qualitatively, i.e., identifying the presence of semantic relations such as paraphrase or elaboration between two texts A and B (Dolan et al. 2004).

The two major approaches in measuring semantic similarity in texts are corpora-based, and knowledge-based (Mihalcea et al. 2006). For instance, one approach for computing the semantic similarity between two texts is to calculate a semantic similarity score directly by representing the texts using a vectorial representation derived, for instance, based on distributional semantics methods which analyze word co-occurrences in large collections of texts. The simplest vectorial representation is the so called Bag of Words (BOW) representation using binary weights, in which case a cosine-based similarity measure leads to a way to quantify word overlap. BOW methods ignore important relationships between words in a sentence such as semantic and syntactic dependencies between words. With the recent advances in deep neural networks, models like Glove (Pennington, Socher, and Manning 2014), ELMO (Kenter and De Rijke 2015) perform much better on many of the NLP tasks. Other semantic similarity algorithms (Taieb, Aouicha, and Hamadou 2014; Zhu et al. 2018) use WordNet, i.e., a lexical database that was manually built, where words are organized in synonymous sets (synsets) which in turn are hierarchically organized in an ontology.

Furthermore, different neural networks such as Recurrent Neural Networks (Mueller and Thyagarajan 2016; Kiros et al. 2015) and Convolutional Neural Networks (CNN) (He, Gimpel, and Lin 2015) provide good performance on sentence level semantic similarity tasks. In addition to this, the introduction of the transformer model, BERT (Devlin et al. 2018) has been able to achieve state-of-the-art results in various NLP tasks. We do report results with BERT.

Related Works

In this section, we review some of the previous work in the area of self-explanations and semantic similarity, with a focus on work related to source code comprehension. Self-explanations are beneficial to learning as during self-explanations several cognitive mechanisms are involved, including creating inferences to fill in gaps in learning materials, integrating new information with old knowledge, and monitoring and fixing defective knowledge (Roy and Chi 2005). Self-explanations can be elicited in a variety of ways, resulting in various types of self-explanations, such as free self-explanations (no specific instruction or training is offered, just simple prompting to explain) or scaffolded self-explanations (these are self-explanations with support from a more knowledgeable other, e.g. a human or computer-based tutor). Self-explanations can have different degrees of impact on various learners, e.g., learners who write down their thoughts when engaging with a specific learning resource, such as reading scientific articles or while attempting to solve a problem, benefit in general from such self-explanations and in particular this type of self-explanation has been proven to be more appropriate for students who struggle with challenging reading activities, such as reading scientific texts that require a higher cognitive load.

A series of studies (Recker and Pirolli 1990; Pirolli and Recker 1994; Bielaczyc, Pirolli, and Brown 1995) focused on the role of self-explanations for programming with positive results, e.g., it helped to understand the concepts of Lisp programming. While a significant relationship was found between skill improvement and the amount of self-explanation generated, they also discovered that the nature of self-explanations is important: when compared to low-performing students, high-performing students' explanations were far more structured. Studies performed with undergraduates (Rezel 2003) and high school students (Alhassan 2017) showed that students using self-explanations were better at program construction compare to those who did not self-explain.

Early attempts on semantic similarity used feature engineering (Gupta and El Maarouf 2014), e.g., they generated 20 linguistic features, which were then used as predictors in a support vector regressor. In (Zhao, Zhu, and Lan 2014), the authors created a system that performed best on task 1 of SemEval2014, the leading forum for semantic evaluations. They used seven textual features in combination with various machine learning algorithms like support vector regressors, random forest regressors, and others. (Maharjan et al. 2017) used an ensemble of traditional machine learning algorithms with deep learning models for Semeval

2017, where their system was one of the top performing system. Sultan (Sultan, Bethard, and Sumner 2015) proposed a supervised architecture model which relies on word alignments and similarities between compositional sentence vectors.

Elvys (Pontes et al. 2018) proposed a neural network architecture that uses siamese CNN for analyzing the context of a word in a sentence and generating its representation and then using siamese LSTM to analyze the whole sentence based on words and the local context. After completing those two steps, the semantic similarity between two sentences is calculated based using a Manhattan distance. Chen et al. (Chen et al. 2017) proposed a method for semantic similarity for web service discovery that integrates multiple conceptual relationships for web service discovery. The similarity is computed as a weighted aggregation of interface similarity and web service description similarity.Vekariya (Vekariya and Limbasiya 2020) introduced a new approach called DeepLSTM for semantic similarity for answer selection in question answering. The approach converts words into vectors using word embedding. Once the words are embedded, a versatile global T-max pooling and DeepLSTM are used to predict an output score.

Experiments and Results

We have conducted experiments with various models (sets of textual features) to tackle the task of automatically evaluating students' self-explanations of computer programs with the specific goal of predicting a semantic similarity score which indicates how semantically similar the self-explanations are to the expert explanations.

Data Collection

All the data were collected using a web-based learning tool called PCEX, which is a collection of interactive worked examples that allow students to explore text explanations of programming code line by line interactively (see Figure 1). Using the PCEX original platform, we made some changes to prompt students to explain each line of code as a way to collect genuine student self-explanations. Specifically, students were shown 10 different JAVA examples which has on average 12 lines of code and asked to read each line of code and explain it which resulted in 1,771 self-explanations. Since for each line of code we also have the benchmark, expert-generated explanations from the original PCEX platform, we ended up with a data-set that consists of 1,771 sentence pairs, a student explanation, and an expert explanation for each line of code.Table 1 shows examples of line-by-line student self-explanations and the corresponding benchmark explanations. Descriptive statistics such as average length of words and standard deviation for student explanations and benchmark explanations are presented in Table 2.

Data Preparation

Human experts annotated the sentence pairs with semantic similarity ratings ranging from 1-5 (1 - not similar, 5 - semantically similar). The goal of the annotation was to generate a gold standard for training and testing the machine learning models that we experimented with.

Student Explanation	Standard Explanation
Declares the array we want to use for our assignment	We initialize the array of type int to hold the specified numbers.
Shift the point by +11 in the x direction and +6 in the y direction.	The translate method receives two parameters.
create variable integer entitled "num" with value 5	In this program, we initialize the variable num to 15.
Initialize the maximum value to the first value in the array.	We need to extract the first letter from the last name.
print the max value after the recursion ends	This statement prints the maximum value of the array to the default standard output stream.

Table 1: Student line-by-line self explanation vs PCEX Standard Explanation

Due to space reasons, we provide only briefly the details of the annotation protocol as in the followings:

- Explanations that contain more than one sentence were broken down into individual sentences, and all pairs of sentences were generated.
- for each sentence in the student self-explanation, find the sentence in the expert explanation that best matches in terms of semantic similarity. Judge the semantic similarity by using a rating from 0 (not similar, which could mean irrelevant or incorrect; well-known misconceptions were flagged) to 5 (similar; the student self-explanation is correct and covers all concepts of the expert explanation).

Explanations	Word Count (μ)	Word Count (σ)
Student	11.862	6.602
Standard	16.815	6.793

Table 2: Statistics of student explanation and standard explanation

To annotate the data, we recruited six graduate students with expertise in computer programming. Each sentence pair was rated by three students, with three students rating the first half of the data and the other three students the other half. The annotation for each sentence pair was conducted in two stages. During the first stage, each of the annotators provided a score for each pair. During the second stage, disagreement cases were discussed, and raters were given a chance to change their original scores. After the second stage, each sentence pair (except very few cases) had similarity ratings which differed by at most 1 point among the three annotators. We considered such a difference of zero or one as "agreement" owing to subtle differences in interpreting the sentences, which was realized during the disagreement resolution stage.

The inter-rater agreement between raters was computed using Fleiss Kappa (Fleiss 1971). Fleiss' Kappa was 0.33 after stage 1, representing the fair agreement between the annotator and 0.99 during stage 2.

Models	R1		R2		R3		R4		R5	
	Corr.	RMSE	Corr.	RMSE	Corr.	RMSE	Corr.	RMSE	Corr.	RMSE
Linear	0.5251	0.2338	0.6792	0.2035	0.4679	0.2445	0.6895	0.2016	0.6655	0.2056
Support Vector	0.4070	0.2518	0.6572	0.2115	0.4471	0.2466	0.7088	0.1995	0.6729	0.2042
Random Forest	0.5241	0.2341	0.6628	0.2059	0.5375	0.2350	0.6652	0.2059	0.6628	0.2059
Decision Tree	0.5033	0.2373	0.6341	0.2119	0.5120	0.2385	0.6341	0.2119	0.6341	0.2132

Table 3: Summary of the results obtained with all runs R1-5.

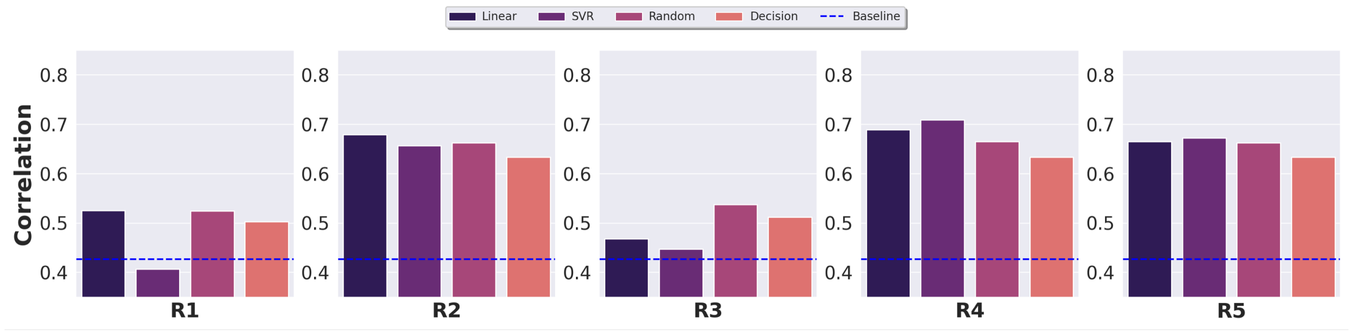


Figure 2: Comparison of models in different runs R1-5 with the baseline

We also normalized the ratings: a normalized score of 0 was assigned for the human rating 1 (meaning not similar at all), a score of 0.25 for human rating 2, and so on. The normalized score of 1 corresponds to the human rating of 5, meaning most similar. To generate one rating score, we took the average of the three human scores for each instance.

The Features Used in The Models

These features were then used in models trained with the following four different regression models: Linear Regression (LR), Support Vector Regression (SVR), Random Forest Regressor (RFR), and Decision Tree Regressor (DTR). The number of overlapping words and the BERT similarity score correlated at the magnitude of 0.5718, whereas the number of overlapping words and the similarity score obtained from the semantic similarity method used in a state-of-the-art intelligent tutoring system has a magnitude of 0.6415 (these are the highest, significant correlations among our features). Based on those correlations, we decided to use the following combinations of features as they were not highly correlated with each other:

- **R1:** Word count difference, Number of overlapping words, Number of bigram overlapping words
- **R2:** Word count difference, Number of bigram overlapping words, Semantic Similarity Score obtained from Sentence BERT pre-trained model
- **R3:** Word count difference, Number of bigram overlapping words, Semantic Similarity Score obtained from DeepTutor
- **R4:** Semantic Similarity Score obtained from Sentence BERT pre-trained model, Semantic Similarity Score obtained from DeepTutor
- **R5:** Semantic Similarity Score obtained from Sentence BERT pre-trained model

Results

We conducted experiments with models that combine different sets of features. We present here the results of the best combinations of features and algorithms (LR, SVR, etc.). For each of the models, we used a 10-fold training-testing methodology in which the dataset was divided into 10equal folds, and each fold was held-out for testing resulting in 200 train-test iterations. We calculated a correlation between the semantic similarity score given by human annotators and the semantic similarity score obtained from the DeepTutor semantic similarity engine, which was trained on Physics data that has a correlation score of 0.4270, which we used as our baseline to compare the models' performance.

Table 3 shows the performance of our model in each runs, and Figure 2 shows the comparison of our model's performance with baseline. Among all runs(R1-R5), SVR on R4 produced the best performance across all the models with a correlation score of 0.7088. Hence for the regression task explored in this paper, the set of features represented by R4 (Semantic similarity score obtained from Sentence BERT pre-trained model, semantic similarity score obtained from previously developed DeepTutor semantic similarity engine) has been the most significant.

We also ran an experiment to understand the performance of various models for each annotation category. Table 4 shows the correlation score for each annotation label for the best performing model. From the table we can infer that the model is performing well for each annotation label especially for annotation label 5 even though this category has fewer instances.

Annotation Label	No. of instances	Corr.
1	520	0.5912
2	513	0.6755
3	416	0.67335
4	263	0.6155
5	59	0.7517

Table 4: Performance of best model per majority human annotation score.

Error Analysis

While the best model provides very good performance compared to human ratings, it does fail in many instances which may represent opportunities for improvement. We conducted an in-depth analysis of the major failing points of the best model as indicated by large differences between the human ratings and the best model's predicted similarity score. Such instances with large discrepancies are shown in Table 5. A closer analysis of those instances revealed that sometimes students provide correct explanations in very different language compared to the benchmark explanation provided by experts (see instance 3 in the table) or the student explanation focuses on a higher level explanation of the corresponding line of code explaining why the line was needed (why?) whereas the benchmark explanation focuses on how the intended step is implemented (see instance 1).

	Student Explanation	Benchmark Explanation
1	Read in the user's input as to whether the phone is broken	The variable isBroken is true when the phone is broken, and false otherwise.
2	creates "translate" with dependent integers dx and dy	This method shifts the coordinates by a specific delta-x and delta-y, which are passed as parameters.
3	this is beginning line of the translate method of the class Point1, it has two integers as its arguments, dx and dy	This method shifts the coordinates by a specific delta-x and delta-y, which are passed as parameters.
4	Creates a function called translate which takes 2 ints called dx and dy	This method shifts the coordinates by a specific delta-x and delta-y, which are passed as parameters.
5	the firstInitial and the lastInitial are then stored together in the variable named initals.	This statements concatenates the extracted initials and store the result in the string initials.

Table 5: Sentence pair instances where the similarity score varied

More domain specific training of semantic representations are needed, e.g., in the form of jointly trained code and text embeddings, to better handle instance 3, e.g., identifying that dx and delta-x refer to the same concept and that 'integers' in the student explanation are equivalent to 'delta-x and delta-y' in the benchmark explanation as they are integer variables. Instances similar to instance 1 can be better handled by making the prompts more specific, e.g., prompting students to explain both the why and how of a line of code as opposed to just asking them to explain which is vague and students may just explain the why or just the how.

Conclusions

In this paper, we evaluated four different regression models, namely, linear regression, support vector regression, random forest regression, and decision tree regression with a number of features in different combinations in order to compute a semantic similarity score between student self-explanations and benchmark explanations. The results obtained indicate that support vector regression (SVR) has the highest performance across all sets of models when used with a combination of features that includes a semantic similarity score obtained from the Sentence BERT pre-trained model and the semantic similarity score obtained from DeepTutor. We plan to extend this work further by trying different state-of-the-art deep learning methods. Also, we plan to mix the hand-crafted features with word embeddings derived using neural networks to create hybrid models.

Acknowledgement

This work has been supported by two NSF awards: the Learner Data Institute (NSF; award 1934745) and CSEdPad: Investigating and Scaffolding Students' Mental Models during Computer Programming Tasks to Improve Learning, Engagement, and Retention (NSF award 1822816). The opinions, findings, and results are solely the authors' and do not reflect those of NSF.

References

Agirre, E.; Banea, C.; Cardie, C.; Cer, D.; Diab, M.; Gonzalez-Agirre, A.; Guo, W.; Lopez-Gazpio, I.; Maritxalar, M.; Mihalcea, R.; et al. 2015. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, 252–263.

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.

Alhassan, R. 2017. The effect of employing self-explanation strategy with worked examples on acquiring computer programing skills. *Journal of Education and Practice* 8(6):186–196.

Bielaczyc, K.; Pirolli, P. L.; and Brown, A. L. 1995. Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and instruction* 13(2):221–252.

Buse, R. P., and Weimer, W. R. 2008. A metric for software readability. In *Proceedings of the 2008 international symposium on Software testing and analysis*, 121–130.

Chen, F.; Lu, C.; Wu, H.; and Li, M. 2017. A semantic similarity measure integrating multiple conceptual relationships for web service discovery. *Expert Systems with Applications* 67:19–31.

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.

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.

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*.

Dolan, W.; Quirk, C.; Brockett, C.; and Dolan, B. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources.

Fleiss, J. L. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76(5):378.

Gupta, R., and El Maarouf, I. 2014. Uow: Nlp techniques developed at the university of wolverhampton for semantic similarity and textual entailment. In *ACL and Dublin City University*. Citeseer.

He, H.; Gimpel, K.; and Lin, J. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, 1576–1586.

Kenter, T., and De Rijke, M. 2015. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, 1411–1420.

Kiros, R.; Zhu, Y.; Salakhutdinov, R. R.; Zemel, R.; Urtasun, R.; Torralba, A.; and Fidler, S. 2015. Skip-thought vectors. In *Advances in neural information processing systems*, 3294–3302.

Maharjan, N.; Banjade, R.; Gautam, D.; Tamang, L. J.; and Rus, V. 2017. Dt_team at semeval-2017 task 1: Semantic similarity using alignments, sentence-level embeddings and gaussian mixture model output. In *Proceedings of the 11th international workshop on semantic evaluation (semeval-2017)*, 120–124.

Mihalcea, R.; Corley, C.; Strapparava, C.; et al. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, 775–780.

Mueller, J., and Thyagarajan, A. 2016. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

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

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Pirolli, P., and Recker, M. 1994. Learning strategies and transfer in the domain of programming. *Cognition and instruction* 12(3):235–275.

Pontes, E. L.; Huet, S.; Linhares, A. C.; and Torres-Moreno, J.-M. 2018. Predicting the semantic textual similarity with siamese cnn and lstm. *arXiv preprint arXiv:1810.10641*.

Recker, M. M., and Pirolli, P. 1990. A model of self-explanation strategies of instructional text and examples in the acquisition of programming skills.

Rezel, E. S. 2003. The effect of training subjects in self-explanation strategies on problem solving success in computer programming.

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.; McCarthy, P. M.; Lintean, M. C.; McNamara, D. S.; and Graesser, A. C. 2008. Paraphrase identification with lexico-syntactic graph subsumption. In *FLAIRS conference*, 201–206.

Rus, V.; Akhuseyinoglu, K.; Chapagain, J.; Tamang, L.; and Brusilovsky, P. 2021. Prompting for free self-explanations promotes better code comprehension.

Schulte, C.; Clear, T.; Taherkhani, A.; Busjahn, T.; and Paterson, J. H. 2010. An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports*. 65–86.

Sultan, M. A.; Bethard, S.; and Sumner, T. 2015. Dls@cu: Sentence similarity from word alignment and semantic vector composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 148–153.

Taieb, M. A. H.; Aouicha, M. B.; and Hamadou, A. B. 2014. Ontology-based approach for measuring semantic similarity. *Engineering Applications of Artificial Intelligence* 36:238–261.

Tamang, L. J.; Alshaikh, Z.; Ait-Khayi, N.; and Rus, V. 2020. The effects of open self-explanation prompting during source code comprehension. In *The Thirty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS-32)*.

Vekariya, D. V., and Limbasiya, N. R. 2020. A novel approach for semantic similarity measurement for high quality answer selection in question answering using deep learning methods. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 518–522. IEEE.

Zhao, J.; Zhu, T.; and Lan, M. 2014. Ecnu: One stone two birds: Ensemble of heterogenous measures for semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, 271–277.

Zhu, X.; Li, F.; Chen, H.; and Peng, Q. 2018. An efficient path computing model for measuring semantic similarity using edge and density. *Knowledge and Information Systems* 55(1):79–111.