

Can LLMs Replace Semantic Similarity for Scoring Student Code Explanations?

Arun-Balajiee Lekshmi-Narayanan,

Mohammad Hassany,

Peter Brusilovsky

University of Pittsburgh,

Correspondence: arl122@pitt.edu

Abstract

In programming courses, students are often asked to explain code fragments, which is a way to assess their understanding of programming constructs and patterns. These types of problem known as “explain in plain English” are valuable in both assessment and practice contexts. The main challenge to using these types of problems at scale in both contexts is automating the scoring process, i.e., assessing whether those explanations are correct. The prevailing approach scores an explanation by its semantic similarity to an instructor’s model explanation, but this raises a measurement concern: students who reason correctly yet phrase their explanations differently from an expert may be scored as incorrect (false negatives), threatening the validity and fairness of the assessment. Given recent advances in LLM-based automated scoring, it remains unclear whether semantic similarity methods are still the most effective technique for automatically scoring free-form student responses, such as code explanations. In this paper, we present a rigorous comparison between LLMs and semantic similarity approaches for the automated scoring of student code explanations, using an open dataset Selfcode 2.0. We frame the scoring as a binary classification task and use generative AI to balance the dataset. Our results suggest that LLM-based scoring ($F1 = 0.98$, accuracy = 0.96) outperform semantic similarity scoring ($F1 = 0.72$, accuracy = 0.65). It also produces fewer false negatives and eliminate the need to produce well-formulated model explanations.

1 Introduction

In programming, the ability to understand and explain code and code fragments has long been considered an important skill. To master this skill, students are encouraged to study multiple worked examples (WE) where the lines or fragments of a code solving a problem are explained by experts (Linn, 1992; Brusilovsky et al., 2008). Sum-

marizing multiple years of research on learning, the ICAP framework (Chi and Wylie, 2014) argues that active learning is more effective than passive learning. An efficient way to turn the code explanation skills from passive to active is to practice explaining the code themselves rather than reading experts’ explanations. (Chi et al., 1989, 1994). Currently, this active approach is used in both assessment and study contexts. In an *assessment context*, instructors include code explanation questions (known as “explain in plain English”) in home assignments and exams. In a *study context*, students are encouraged to self-explain code-based WE either by adding explanatory comments to WE code (Garces et al., 2023) or by writing code explanations to their workbooks (Vihavainen et al., 2015).

Evaluating student-written explanations is a hard task for instructors, both in terms of grading and providing corrective feedback. As a result, such questions are rarely assigned in take-home assessments, and when they do appear in study contexts, the evaluations are often neglected entirely. This leaves the learning process incomplete: students need proper feedback on their written explanations so that they do not form dangerous misconceptions (Kerslake et al., 2025). Good automatic assessment systems could help fill this gap. Advances in natural language processing (NLP) have enabled automatic evaluation of student self-explanations by comparing them to expert explanations using semantic similarity by representing the sentences using vector embeddings (Reimers and Gurevych, 2019; Chapagain and Tamang, 2022; Oli et al., 2024b). More recently, LLM-as-a-judge has been explored as a potentially more direct alternative to semantic similarity. In a practice context, (Oli et al., 2024a) investigated a fine-tuned open-source LLM with preference optimization to generate feedback for students in a dialog-based intelligent tutoring system, while (Oli et al., 2024c) used prompt engineering to assess LLMs’ ability to identify gaps

and student misconceptions when comprehending code. In an assessment context, (Denny et al., 2024; Kerslake et al., 2025) used LLM-based approaches to score problems that assessed student comprehension skills by asking them to explain the code of small programs. However, none of these pioneering efforts compared semantic similarity with LLM-powered automated assessment.

A problem with reliably comparing the effectiveness of LLMs for automatically evaluating students’ responses using semantic similarity methods is the paucity of high-quality task-specific datasets. Our work presented in this paper was enabled by a recently published dataset of labeled code explanations SelfCode 2.0 (Chapagain et al., 2025), which contains 1854 pairs of labeled student explanations along with corresponding expert explanations. To use this dataset, however, we had to overcome the imbalanced nature of this dataset: 1794 student explanations in the dataset are labeled as correct and only 60 as incorrect, with a ratio of positive to negative samples of approximately 30:1. Traditional oversampling techniques such as SMOTE are commonly used to address class imbalance but require numerical feature representations.

Our work follows this trend as the first study to use synthetically generated negative examples to balance an educational dataset, specifically to compare semantic similarity and LLM-based approaches to automated assessment of student code explanations. We frame the automated assessment of student code self-explanations as a measurement problem: an automated scorer should yield judgments of explanation correctness that are *valid*, *reliable* across items and raters, and *fair* to students whose correct explanations differ in wording from an expert reference. Semantic similarity methods, which score an explanation by its closeness to a single expert explanation, risk misclassifying such students as incorrect (false negatives). We investigate whether prompting an LLM as a judge yields more dependable correctness judgments than semantic similarity, without requiring a ground-truth explanation or a tuned decision threshold. Using five-fold cross-validation with statistical tests appropriate for a small number of folds, we find that LLM-based scoring ($F1 = 0.98$) outperforms the strongest semantic similarity baseline ($F1 = 0.72$), with markedly fewer false negatives. These results suggest that LLM-as-a-judge is a viable, threshold-free alternative for assessing student explanations, with potential fairness benefits for stu-

dents who reason correctly but express themselves differently from experts.

2 Method

2.1 Dataset

To answer our research question, we used the recently released SelfCode2.0 dataset (Chapagain et al., 2025), which includes explanations produced by 60 students and 2 experts for individual lines of four code examples. In the dataset, each student’s explanation is labeled by annotators as correct or incorrect. In total, the dataset included 3019 pairs of student and expert explanations for the same code lines. For our study, we used a subset of these pairs (1854 out of 3019) in which both a student and an expert explained the line using a single statement. This selection was made to facilitate the exploration of modern semantic similarity approaches that focus on complete sentences. In these 1854 pairs, 1794 student explanations were labeled as correct (positive examples) and 60 as incorrect negative examples. Our secondary contribution in this work addresses the class imbalance in the dataset by synthetically generating negative examples using LLMs.

2.2 LLM Based Evaluation

Our LLM-based approaches for evaluating student explanations are based on treating an LLM-as-a-Judge that directly predicts the *correctness* of student explanations. We prompt the LLM with the context that includes the problem statement of the explained WE, the code line to explain along with the line number, and the associated student explanations from the dataset (Figure 1 in Appendix). In all prompts, we use the same section headings in capital letters, such as “PROGRAM DESCRIPTION”, “LINE NUMBER” and so on. We used the prompts with GPT-3.5-Turbo-16k for the automated assessment task.

No Definition (LLM-NoDef) We did not define what we mean by “correctness,” relying instead on what the LLM itself means by that.

Construction (LLM-Constr) We defined correctness by focusing on the construction-based approach to explain a code line, i.e., the role of the line in solving the programming problem.

Code Behavior (LLM-Behavior) We defined correctness by focusing on the behavior-based ap-

proach to explain a code line, i.e., the result of executing this line.

2.3 LLM-as-a-Judge Prompt

Before finalizing the prompt, we manually reviewed 200 student explanations across the similarity range. We observed that some students focused on explaining the line’s behavior (behavior-focused explanations). In contrast, others concentrated on why the line was used to achieve the program’s goal (construction-focused explanations). To avoid unfairly assessing students who favored one of these explanation styles, we developed 3 versions of the LLM prompt, each with a different definition of correctness at the beginning.

Determine the correctness of the student’s explanation based on a given source code, specific line number.

(a) Set "Correctness" to 1 if the student’s explanation is correct and 0 if it is incorrect.
OR
(b) "Correctness" A correct EXPLANATION explains why the line is used while implementing this program given the PROGRAM DESCRIPTION and SOURCE CODE below.
OR
(c) "Correctness" A correct STUDENT EXPLANATION explains the code behavior using the code syntax. ONLY set "Correctness". DO NOT PROVIDE REASON.

PROGRAM DESCRIPTION
{program desc}
SOURCE CODE
{source code}
LINE NUMBER
{line num}
STUDENT EXPLANATION
{student explanation}
CODE
{line content}
Correctness:

Figure 1: Prompting LLM to evaluate students’ explanations. “OR” notation is used in this to indicate only one of the 3 possibilities were used to define “Correctness” to the model (a) = No Definition, (b) = Construction, (c) = Code Behavior Prompt

2.4 Semantic Similarity Based Evaluation

We used three popular versions of semantic modeling approaches:

Deep-Tutor (DT) toolkit that adopted the algorithm described to evaluate correctness in the DT-GRADE corpus (Banjade et al., 2016) and was used in process of data collection of our dataset (Chapagain et al., 2025). The method generates similarity

scores using Support Vector Regression with a feature set combining SentenceBERT and DeepTutor similarity scores (Rus et al., 2013).

RoBERTa (Liu et al., 2019) a version of BERT with appropriate modifications for Next Sentence Prediction tasks, available through sentence-transformers¹.

GPT Sentence Embeddings (GPT-SE)² available as an API that creates a vector representation for any form of text, either broken down into sentences, individual tokens, or entire paragraphs using the text-embedding-3-small from OpenAI.

For each approach, we computed semantic similarity scores ranging from 0 to 1 for each pair of student and expert explanations. Deep-Tutor is designed to return this score directly. To obtain a semantic similarity score using RoBERTa or GPT-SE, we generated sentence embeddings for all student and expert explanations. We calculated cosine similarity between the embeddings of student and expert explanations within each pair.

2.5 Comparing Semantic Similarity and LLM-Based Evaluation on a Balanced Dataset

To address the class imbalance in the original dataset, we created a balanced dataset with 1794 positive and negative examples. To generate negative examples, we used a small language model (SLM) GPT-OSS 20b (OpenAI, 2025) with the prompt shown in Figure 2. We chose the model because it differs from the one we use to assess student explanations. We selected the lines of code for which student explanations are available in the SelfCode2.0 dataset (Chapagain et al., 2025) and prompted the SLM to generate 3 distinct incorrect explanations for every line of code.

To verify whether the GPT-OSS 20b generated negative explanations that were indeed incorrect, we sampled 100 explanations from the dataset and 2 researchers from the team independently annotated and confirmed that at least 95% of the sample had incorrect explanations with an agreement of 91%. The annotated samples from the 2 annotators are available for review on Drive³. One of the annotators further confirmed that an additional

¹<https://huggingface.co/sentence-transformers/all-roberta-large-v1>

²<https://platform.openai.com/docs/guides/embeddings>

³<https://docs.google.com/spreadsheets/d/102WoJ1Y1EhtCIQcuWoGxav1HAkqsqMSE6a-ZdtfmXUw/edit?usp=sharing>

100 explanations ⁴ also had at least 95% incorrect explanations.

```
# Task Generate **3 Incorrect STUDENT EXPLANATIONS** for the specified **LINE NUMBER** in the
given **SOURCE CODE**.
### Correctness
- The **STUDENT EXPLANATION** must accurately
describe the behavior of the specified line of code. - It
should correctly reference syntax, logic, and execution
details of the **SOURCE CODE**.
—
## Provided Information
**Program Description:**
program_desc
**Source Code:**
source_code
**Line Number:**
line_num
**Code:**
line_content
**Incorrect Student Explanations:**
student_explanation
**Expert Explanation:**
expert_explanation
— ## Output Format
Return the final answer **only in valid JSON** with
the following structure:
{
  "incorrectExplanations": [ "explanation 1",
    "explanation 2",
    "explanation 3" ]
}
```

Figure 2: Artificial Generation to augment the original dataset with negative (incorrect) code examples

We conducted 5-fold cross-validation on the balanced dataset. The training set in every fold consisted of 1435 positive and negative examples each, and the testing sets consisted of 359 positive and negative examples each. We used the training set to identify the best threshold for the semantic similarity baselines using the optimal F1 scores. In contrast, we used the test sets to compare LLM performance and to assess student explanations using semantic similarity. The LLMs were not trained on the training set, since we were using them in a zero-shot setting.

2.6 Statistical Analysis

We treated the per-fold F1 scores from the 5-fold cross-validation as paired observations across the six approaches. Because cross-validation training folds overlap substantially, the independence assumption underlying the ordinary paired *t*-test is violated and its Type I error is inflated (Dietterich,

1998; Nadeau and Bengio, 2003); we therefore used non-parametric and variance-corrected procedures throughout. We first tested for an overall difference among methods with the Friedman test, reporting the Iman–Davenport *F* statistic, which is preferred over the χ^2 approximation when the number of blocks is small (Iman and Davenport, 1980; Demšar, 2006), together with Kendall’s *W* as a measure of agreement among the per-fold rankings. For pairwise comparisons, we used the variance-corrected paired *t*-test of Nadeau and Bengio (2003), which rescales the variance by $(1k + n_{\text{test}}n_{\text{train}})$ to account for the dependence between overlapping folds, and we report the mean F1 difference with its 95% confidence interval. As a rank-based post-hoc, we applied the Nemenyi test (Demšar, 2006); with only five folds its critical difference is large, so it certifies only the most rank-separated pairs. Given the small number of folds, we foreground effect magnitudes (mean differences and confidence intervals) alongside statistical significance.

3 Results

As the Table 2 shows, the LLM Behavior (F1 = 0.98, Accuracy = 0.96) outperforms Deep Tutor (F1 = 0.69, Accuracy = 0.6) and RoBERTa (F1 = 0.7, Accuracy = 0.61) (see Table 2). The best performance on 5-fold test sets of LLM Behavior ($TP=357, FP=5, FN=1, TN=2$) was also better than Deep Tutor’s ($TP=341, FP=317, FN=18, TN=42$), where TP = True Positives, FP = False Positives, FN = False Negatives and TN = True Negatives. More results are available on Drive ⁵ and the Table 1. Fewer false negatives for LLM-powered automated scoring suggests that the system evaluates a smaller number of correct student explanations as incorrect. Fewer false negatives makes the system less frustrating for the students to work with when they are assessed incorrect for cases that the students know their answers are correct.

The LLM-Constr. potentially underperformed (for both datasets – F1 = 0.89, Acc = 0.8) because the student explanations in the dataset described the “syntax and code behavior” and missed out the context of the code within the rest of the program, indicating that the prompts to the LLMs must balance the rubrics to assess students’ explanations between the instructors’ expectations and students’

⁴<https://drive.google.com/file/d/135043Q2XexjrNG6XBriuEHjBnwe3eIA3/view?usp=sharing>

⁵<https://docs.google.com/spreadsheets/d/1qj5uz1AWwEIkXm8mn1BA-IHeILYulIq4ZofL14pvozs/edit?usp=sharing>

Method	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Deep Tutor	0.67	0.69	0.70	0.70	0.69
RoBERTa	0.69	0.71	0.69	0.72	0.71
GPT-SE	0.68	0.72	0.71	0.75	0.72
LLM-Behavior	0.98	0.98	0.98	0.98	0.99
LLM-NoDef	0.98	0.98	0.97	0.98	0.99
LLM-Constr	0.88	0.89	0.87	0.91	0.89

Table 1: Per-fold F1 across the five cross-validation folds. Values are mean $\pm$ SD across folds.

Model	F1 (mean $\pm$ SD)	Acc. (mean $\pm$ SD)
Deep Tutor	0.69 $\pm$ 0.01	0.60 $\pm$ 0.03
RoBERTa	0.70 $\pm$ 0.01	0.61 $\pm$ 0.02
GPT-SE	0.72 $\pm$ 0.02	0.65 $\pm$ 0.04
LLM-Behavior	0.98 $\pm$ 0.01	0.96 $\pm$ 0.01
LLM-NoDef.	0.98 $\pm$ 0.01	0.96 $\pm$ 0.01
LLM-Constr.	0.89 $\pm$ 0.02	0.80 $\pm$ 0.02

Table 2: Comparing Semantic Similarity and LLM approaches on test sets in a 5-fold CV setup with a balanced dataset using artificial data. Values are mean $\pm$ standard deviation across the five folds.

understanding of a correct explanation for a line of code. The performance of the similarity-based methods being lower than the LLMs could suggest that LLM methods are more robust than similarity-based methods when considering zero-shot performance with educational assessment tasks.

3.1 Statistical Comparison

The six approaches differed significantly overall (Friedman $\chi^2(5) = 23.97$, $p < .001$; Iman-Davenport $F(5, 20) = 93.22$, $p < .001$), and the near-ceiling Kendall’s $W = 0.96$ indicates that the ordering of methods was almost perfectly consistent across folds. Variance-corrected paired t -tests (Table 3) confirmed that every LLM prompt outperformed every semantic similarity baseline. The strongest baseline, GPT-SE, was exceeded by LLM-Behavior by a mean F1 of 0.265 (95% CI [0.225, 0.306]), with comparable margins over RoBERTa and Deep Tutor. Among the prompts, LLM-Behavior and LLM-NoDef both significantly outperformed LLM-Constr ($p \leq .002$, Holm-corrected), whereas the two leading prompts were statistically close and practically equivalent (mean F1 difference 0.004), suggesting that performance was robust to the choice between the Behavior and No-Definition rubrics. The Nemenyi post-hoc, which uses rank order alone, separated only the most distant pairs (Deep Tutor and RoBERTa vs. LLM-Behavior, and Deep Tutor vs. LLM-NoDef;

Comparison	$\Delta F1$ [95% CI]	$t(4)$	p
LLM-Behavior vs. GPT-SE	0.265 [0.225, 0.306]	18.08	$< .001$
LLM-Behavior vs. RoBERTa	0.278 [0.255, 0.300]	34.55	$< .001$
LLM-Behavior vs. Deep Tutor	0.293 [0.269, 0.316]	34.62	$< .001$
LLM-NoDef vs. GPT-SE	0.261 [0.220, 0.303]	17.38	$< .001$
LLM-Constr vs. GPT-SE	0.172 [0.144, 0.200]	16.95	$< .001$
LLM-Behavior vs. LLM-Constr	0.094 [0.066, 0.122]	9.26	.002
LLM-NoDef vs. LLM-Constr	0.089 [0.060, 0.119]	8.51	.002
LLM-Behavior vs. LLM-NoDef	0.004 [0.001, 0.007]	4.37	.012

Table 3: Pairwise comparisons via the variance-corrected paired t -test (Nadeau and Bengio, 2003) on per-fold F1; p -values Holm-corrected across the eight contrasts. $\Delta F1$ is the mean difference across the five folds.

critical difference = 3.37); its inability to certify closer pairs such as LLM-Behavior vs. GPT-SE, despite a large F1 gap, reflects the limited resolution of rank-based tests at five folds and motivates our reliance on the variance-corrected t -test for the principal comparisons.

4 Discussion

We placed the three semantic similarity approaches in the most favorable position by tuning their decision thresholds on the training folds of the 5-fold cross-validation; even so, all three were outperformed by the two leading LLM-based approaches. This gap was not an artifact of fold-to-fold variation: the rankings were near-perfectly consistent across folds (Kendall’s $W = 0.96$), the overall difference was significant (Friedman $\chi^2(5) = 23.97$, $p < .001$), and each LLM prompt exceeded the strongest similarity baseline by a wide, tightly estimated margin on the variance-corrected paired t -test (e.g., $\Delta F1 = 0.27$, 95% CI [0.23, 0.31]). Interpreted as a measurement problem, these results suggest that LLM-based scoring provides more reliable correctness judgments than semantic similarity for this task, for two reasons: it operates without a ground-truth expert explanation and without a tuned decision threshold, removing two sources of construct-irrelevant variance. Just as importantly, the LLM approaches produced markedly fewer false negatives, indicating fairer treatment of students who explain code correctly but in terms that diverge from an expert reference.

We placed the three evaluated semantic similarity approaches in the most favorable position by finding the optimal threshold using training sets of 5-fold cross-validation. However, when compared to LLM-based approaches on the same dataset, all semantic similarity approaches performed worse than the two best-performing LLM-

based approaches. Our results suggest that LLMs could be an effective replacement for similarity-based methods for 2 reasons: they can operate without a ground-truth response and do not require a threshold to determine the correctness of student explanations. By defining rubrics, such as with Code Behavior Prompt, we achieved strong results with LLMs that are not state-of-the-art (like GPT-3.5) yet still beat contemporary semantic similarity methods. The problem with using similarity-based approaches is that the choice of threshold plays a key role in the final prediction of student correctness. Using nonlinear LLM approaches allows us to capture more information and features from the student explanations that are not categorized solely as lexical or semantic, but instead as latent-space representations that could have meaning or intent similar to that of the expert.

5 Conclusion

In this work, we presented experiments that compared semantic similarity with LLM methods. The LLM methods achieved top F1 and Accuracy scores without threshold tuning or fine-tuning, and better accommodated diverse student writing styles.

Framed as a measurement problem, these findings suggest that LLM-as-a-judge offers a more reliable and fairer basis for scoring student code explanations than semantic similarity: it requires neither a ground-truth reference nor a tuned threshold, and it produces fewer false negatives for students whose explanations are correct but diverge from an expert's. More broadly, our use of generative AI to synthesize negative examples offers educational measurement researchers a practical template for constructing balanced evaluation sets where labeled data are scarce.

6 Limitations

Our work is limited to assessing the LLM-powered assessment for student explanations. As future work, we could compare our methods across more datasets to support comparisons between semantic similarity and LLM-powered assessments, and to evaluate other student short-answer formats (Li et al., 2025; Liu et al., 2025).

A limitation of the original dataset (Chapagain et al., 2025) is the variation among human experts in their assessments of student explanations. Dis-course level (Chen et al., 2025) analyses could sim-

plify and diversify the annotation of texts, which we will investigate further within the context of code explanations.

We generated synthetic negative explanations using LLMs and briefly verified with 2 annotators using a sample of 100 explanations. For a more extensive evaluation we could compare synthetically generated and student-generated explanations pedagogically by focusing on student misconceptions or gaps (Oli et al., 2024c).

We did not employ systematic prompt optimization techniques such as those proposed by (Das et al., 2025; Khattab et al., 2023). Incorporating such methods — along with more clearly defined rubrics (Zhou et al., 2022; ?), which are effective — could further improve LLM-based assessment performance. Prompt optimization both for the generation and assessment of explanations would be an interesting extension of this work.

We did not use methods such as GRPO, recently explored for educational applications (Duan et al., 2026). Preference alignment could be used to align LLM reasoning more closely with expert assessments and to generate synthetic incorrect explanations that better reflect authentic student misconceptions and fall within the distribution of typical student errors.

We did not explore our LLM method with other programming languages, targeted at higher levels of student learning (Biggs and Collis, 2014). We also did not explore extensions to LLM-based assessment beyond correctness classification to generate targeted feedback for students.

Our statistical comparison rests on five-fold cross-validation, which yields only five paired estimates per method. With so few folds, rank-based post-hoc tests such as Nemenyi have limited resolution and may fail to separate methods even when their performance differs substantially, and the paired Wilcoxon signed-rank test cannot reach significance at conventional levels; we therefore relied on the variance-corrected paired t -test of Nadeau and Bengio (2003) for pairwise contrasts and report effect sizes and confidence intervals alongside significance. Repeated or higher-fold cross-validation would supply more paired observations and greater power for all of these tests, and is a straightforward extension.

Acknowledgements

Thanks to Dr. Xiang Lorraine Li for contributions to this work. This work was supported partially by the Grad Student Provost Fellowship of the Intelligent Systems Program at the University of Pittsburgh and the NSF Award Number 1822752.

References

- Rajendra Banjade, Nabin Maharjan, Nobal Bikram Niraula, Dipesh Gautam, Borhan Samei, and Vasile Rus. 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*, pages 182–187.
- John B Biggs and Kevin F Collis. 2014. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press.
- Peter Brusilovsky, I-Han Hsiao, and Michael Yudelson. 2008. Annotated program examples as first class objects in an educational digital library. In *Joint Conference on Digital Libraries, JCDL 2008*, pages 337–340.
- Jeevan Chapagain, Arun Balajiee Lekshmi Narayanan, Kamil Akhuseyinoglu, Peter Brusilovsky, and Vasile Rus. 2025. [SelfCode 2.0: An annotated corpus of student and expert line-by-line explanations of code examples for automated assessment](#). *The International FLAIRS Conference Proceedings*, 38(1).
- Jeevan Chapagain and Lasang Tamang. 2022. Automated assessment of student self-explanation during source code comprehension. In *The international FLAIRS conference proceedings*, volume 35.
- Beiduo Chen, Yang Janet Liu, Anna Korhonen, and Barbara Plank. 2025. [Threading the needle: Reweaving chain-of-thought reasoning to explain human label variation](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33111–33135, Suzhou, China. Association for Computational Linguistics.
- Micheline T. H. Chi, Nicholas De Leeuw, Mei-Hung Chiu, and Christian Lavancher. 1994. Eliciting self-explanations improves understanding. *Cognitive Science Volume 18, Issue 3, July–September, Pages*, 18(3):439–477.
- Micheline TH Chi and Ruth Wylie. 2014. The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist*, 49(4):219–243.
- Micheline T. H. Chi, Miriam Bassok, Matthew W. Lewis, Peter Reimann, and Robert Glaser. 1989. Self-explanations: How students study and use examples to solve problems. *Cognitive Science*, 13:145–182.
- Sarkar Snigdha Sarathi Das, Ryo Kamoi, Bo Pang, Yusen Zhang, Caiming Xiong, and Rui Zhang. 2025. Greater: Gradients over reasoning makes smaller language models strong prompt optimizers. In *The Thirteenth International Conference on Learning Representations*.
- Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.
- Paul Denny, David H. Smith, Max Fowler, James Prather, Brett A. Becker, and Juho Leinonen. 2024. [Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills](#). In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024*, page 283–289, New York, NY, USA. Association for Computing Machinery.
- Thomas G. Dietterich. 1998. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.
- Zhangqi Duan, Nigel Fernandez, and Andrew Lan. 2026. Kaser: Knowledge-aligned student error simulator for open-ended coding tasks. *arXiv preprint arXiv:2601.06633*.
- Sebastian Garces, Camilo Vieira, Guity Ravai, and Alejandra J. Magana. 2023. Engaging students in active exploration of programming worked examples. *Education and Information Technologies*, 28:2869–2886.
- Ronald L. Iman and James M. Davenport. 1980. Approximations of the critical region of the friedman statistic. *Communications in Statistics–Theory and Methods*, 9(6):571–595.
- Chris Kerslake, Paul Denny, David H Smith IV, Juho Leinonen, Stephen MacNeil, Andrew Luxton-Reilly, and Brett A Becker. 2025. Exploring student reactions to LLM-generated feedback on explain in plain english problems. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 575–581.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.
- Tianwen Li, Michelle Hong, Lindsay Clare Matsumura, Elaine Lin Wang, Diane Litman, Zhexiong Liu, and Richard Correnti. 2025. [Chain-of-thought prompting for automated evaluation of revision patterns in young student writing](#). In *Proceedings of the Artificial Intelligence in Measurement and Education Conference (AIME-Con): Works in Progress*, pages 49–65, Pittsburgh, Pennsylvania. National Council on Measurement in Education (NCME).

- Marcia Linn. 1992. Can experts' explanations help students develop program design skills. *International Journal on the Man-Machine Studies*, 36:511–551.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Zhexiong Liu, Diane Litman, Elaine L Wang, Tianwen Li, Mason Gobat, Lindsay Clare Matsumura, and Richard Correnti. 2025. [eRevise+RF: A writing evaluation system for assessing student essay revisions and providing formative feedback](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations)*, pages 173–190, Albuquerque, New Mexico. Association for Computational Linguistics.
- Claude Nadeau and Yoshua Bengio. 2003. Inference for the generalization error. *Machine Learning*, 52(3):239–281.
- Priti Oli, Rabin Banjade, Jeevan Chapagain, and Vasile Rus. 2024a. [Automated assessment of students' code comprehension using LLM](#). In *Proceedings of the 2024 AAAI Conference on Artificial Intelligence*, volume 257 of *Proceedings of Machine Learning Research*, pages 118–128. PMLR.
- Priti Oli, Rabin Banjade, Arun Balajiee Lekshmi Narayanan, Peter Brusilovsky, and Vasile Rus. 2024b. Exploring the effectiveness of reading vs. tutoring for enhancing code comprehension for novices. In *Proceedings of the 39th ACM/SIGAPP symposium on applied computing*, pages 38–47.
- Priti Oli, Rabin Banjade, Andrew M Olney, and Vasile Rus. 2024c. Can LLMs identify gaps and misconceptions in students' code explanations? *arXiv preprint arXiv:2501.10365*.
- OpenAI. 2025. [gpt-oss-120b & gpt-oss-20b model card](#). Preprint, arXiv:2508.10925.
- Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. 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)*, pages 3982–3992.
- Vasile Rus, Sidney D'Mello, Xiangen Hu, and Arthur Graesser. 2013. Recent advances in conversational intelligent tutoring systems. *AI magazine*, 34(3):42–54.
- Arto Vihavainen, Craig S. Miller, and Amber Settle. 2015. Benefits of self-explanation in introductory programming. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, pages 284–289. ACM Press.
- Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. In *The 11th International Conference on Learning Representations*.