# Self-Explanation Tutor for Active Study of CS1 Worked Examples

Arun-Balajiee Lekshmi-Narayanan
arl122@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

Mohammad Hassany
moh70@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

Kamil Akhuseyinoglu
kakhusey@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Rully Hendrawan
rah225@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

Peter Brusilovsky
peterb@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA



## Abstract

Worked examples are a important part of introductory programming, but reading their expert explanations is passive. Self explanation, students explaining the problem and its solution to themselves with subgoal level analysis, turns that study into an active task, yet it is hard to scale because assessing free-text explanations and returning timely feedback has had no easy automated solution. We investigate whether a large language model (LLM) can fill that gap. We build a self-explanation tutor for introductory programming, ESSE, in which students explain lines of worked examples and receive immediate LLM feedback on the correctness and completeness of each explanation, and we pursue two goals. First, we ask whether the LLM judges student explanations well enough to serve as the engine of the tutor; we assess its judgments against two independent human reference standards of different kinds, a single domain expert and a crowd of non-expert raters, each with its own strengths and weaknesses, characterizing both where the LLM is reliable and the systematic tendencies in how it diverges. Second, we ask whether the LLM-based tutoring benefits students; deploying it in an introductory Java course, we find that its feedback leads students to persist and revise rather than abandon a line, that their explanations grow more complete and conceptually richer across attempts, and that students show evidence of learning. These indicate that LLM-based assessment is good enough to power a self-explanation tutor, and that the tutor positively shapes how students study worked examples.

## CCS Concepts

• **Applied computing** → **Interactive learning environments**; • **Human-centered computing** → **HCI theory, concepts and models**; • **Computing methodologies** → **Cross-validation**; **Discourse, dialogue and pragmatics**.


*Conference acronym 'XX, Woodstock, NY*

ACM ISBN 978-1-4503-XXXX-X/2026/06
https://doi.org/XXXXXXX.XXXXXXX

## Keywords

Programming, Self–Explanations, Evaluation, Feedback, LLMs

**ACM Reference Format:**
Arun-Balajiee Lekshmi-Narayanan, Mohammad Hassany, Kamil Akhuseyinoglu, Rully Hendrawan, and Peter Brusilovsky. 2026. Self-Explanation Tutor for Active Study of CS1 Worked Examples. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

In programming education, worked examples present complete solutions whose lines are augmented with expert explanations, helping novices acquire problem-solving schemas before they write code on their own [4, 14, 21]. Reading or watching such explanations, however, is *passive*: the ICAP framework [9] argues that learning deepens as activity moves from passive to active and constructive engagement. A way to make worked-example study constructive is *self-explanation*, in which students articulate the purpose and behavior of each step in their own words [5, 36]. Self-explanation of code is tied to gains in program comprehension and writing skill [24], yet existing implementations share a practical bottleneck: there is no scalable, reliable, and timely way to grade the free-text explanations students produce and to give them feedback. Prior work has students explain code in their own words but largely stops at eliciting or analyzing those explanations [17]; what has been missing is a tutor that automatically assesses each self-explanation and returns that assessment so the student can revise.

The bottleneck is challenging because correct student explanations vary enormously in wording. Approaches that score an explanation by its *semantic similarity* to a single expert reference penalize correct answers phrased unlike the expert [29, 34]. Recent work shows that prompting an LLM to act as a judge of student explanations can match or exceed fine-tuned similarity models [25], opening the door to feedback that does not require an exact reference answer. This makes it feasible to build an interface around the active self-explanation task-one that scores each explanation and returns the score to the student in real time-and to ask what such an interface affords once students use it.

This paper presents that interface and a pilot deployment of it in a real introductory Java course. The tool is itself a pedagogical intervention: it converts the passive act of reading expert explanations

into an active writing task and surrounds that task with layered, immediate feedback designed to keep students productively engaged rather than stuck. For each selected line it returns a judgment of *correctness* (binary) and *completeness* (continuous) together with counts of the concepts present and absent. We study the interface we built around two research questions:

**RQ1** To what extent can an LLM judge the correctness and completeness of student self-explanations-well enough to support a tutor-style interaction that assesses each explanation and returns that assessment to the student?

**RQ2** What impact does the tutor have on students' self-explanation behavior as they study worked examples-in particular, do their explanations improve as they attempt more?

**Contributions.** (1) We present an interactive self-explanation tool for worked examples that delivers immediate, concept-level LLM feedback, and report a pilot of it in CS1. (2) On live CS1 data we characterize *how* the LLM and a human disagree-high raw agreement but a systematic under-scoring bias-and show why a single agreement statistic is misleading under skewed label distributions. (3) We corroborate this against a second, independent human standard-a reliability-filtered crowd-using prevalence-robust statistics suited to skewed labels. (4) We show that explanation *completeness*, and specifically its conceptual content rather than its length, is the signal that tracks student revision and improvement.

## 2 Related Work

### 2.1 Self-Explanation and Active Study of Worked Examples

Self-explanation is a robust learning strategy across domains [8], and within the ICAP framework [9] it is a constructive activity that outperforms passive review. In programming specifically, prompting students to explain code has been integrated into tutors for SQL [36] and Python [5], both reporting comprehension gains, and self-explanation has been contrasted with reading worked examples and code comments [28]. Example systems such as PCEX [15] let students step through worked examples line by line, but reading expert explanations remains passive.

While computer science was one of the first domains where the power of self-explanations was explored, active research of using self-explanations in this field started very recently. So far, self-explanations in this area explored several ideas - explaining programming concepts when solving a problem [12], explaining fragments of code examples [12, 33, 39], and explaining preparatory materials [37, 40, 41]. These experiments brough positive results. For example, the study [39], reported that the Socratic method of guided (scaffolded) self-explanation is more effective than free self-explanation in teaching code comprehension skills among students enrolled in an introductory computer science course. A classroom study exploring self-explanations of educational videos in the context of a database course has found correlations between self-explanations and student understanding, as well as classroom performance [37].

The main technical challenge in deploying self-explanation approaches in computer science courses is developing reliable approaches for assessing self explanations. Scalable tools for evaluating student explanations in the context of learning from worked examples used a simple menu-based approach [10, 12]. However, recent research demonstrated that modern semantic similarity approaches could be used to support an efficient dialog with students in computer science education [33, 35]. In our work, we bring together previous experience in supporting self-explanation in the domain of computer science education [3, 12] with automatic assessment and a scaffolded dialog based on recent semantic similarity research.

### 2.2 Automated Assessment of Explanations and LLM-as-a-Judge

Automatically evaluating free-form student responses has a long history in intelligent tutoring, beginning with semantic-similarity comparison against an instructor's model answer. Latent Semantic Analysis, introduced in AutoTutor, was the first such technique [13, 42], and the SEMILAR toolkit later made semantic similarity a standard tool for scoring student responses [1, 34, 35]. Deep-learning successors replaced hand-built similarity with sentence embeddings, for example a transformer extension of SEMILAR built on Sentence-BERT [7, 32]. All of these methods share a structural limitation: because they score an explanation by its proximity to an expert reference, they reward students who echo the expert's wording over those who express the same idea differently [18], and they reveal little about *which* concepts an explanation is missing-the information most useful for feedback. The same reference-matching bias affects general-purpose overlap and embedding metrics such as BLEU, ROUGE, and BERTScore [20, 30, 43].

Large language models let a grader assess an explanation directly, reasoning about the concepts it should contain rather than matching a reference. Prompted with few-shot examples and chain-of-thought reasoning, an LLM can judge the correctness of a line-level explanation as well as or better than fine-tuned similarity models [25, 29], and this *LLM-as-a-judge* paradigm has grown rapidly [19].

Closest to our work are recent systems that use LLMs to assess code self-explanations. An open-source LLM was fine-tuned with preference optimization and embedded in a dialogue-based tutor in which students make repeated attempts to explain a line of code [26, 27]. Explanation correctness has also been assessed indirectly by submitting a student's explanation to an LLM as a specification, generating code from it, and checking whether that code behaves like the original [11]. We take a more direct route: we prompt an LLM to judge each line-level explanation's correctness and completeness and to report the concepts it covers, without fine-tuning and without matching an expert reference. We then benchmark this judge against a semantic-similarity baseline on a public dataset [6], audit its judgments against human raters on live data, and use its concept-level output to drive feedback.

## 3 ESSE: An LLM-Powered Self-Explanation Tutor

We developed a self-explanaton tutor ESSE (*Example Study with Self-Explanations*) (ESSE), a new type of interactive "smart content" that can be used on its own or 4 23, embedded in a any personalized practice portal such as Mastery Grids [23]. ESSE replaces the passive "read the expert explanation" step of a worked-example viewer with an active "write your own explanation" step, then returns immediate feedback (Figure 1). The design goal is pedagogical: move the student from passive reading toward constructive articulation, in the sense of ICAP [9], while keeping the cost of getting unstuck low. In placing an LLM at the center of assessment and feedback, ESSE follows recent generative intelligent tutoring systems [22], but targets line-level self-explanation of worked examples.

### 3.1 The Self-Explanation Activity

A student works through a worked example one line at a time. For each highlighted line, ESSE asks the student to explain, in their own words, why that line is used while constructing the program given the goal description, and to submit the explanation. The first attempt is unscaffolded. On submission, ESSE evaluates the explanation and returns feedback; the student may revise and resubmit as often as they like, and the control to advance to the next line becomes available once the explanation is judged correct. The tutor's targets are explanations that are both *correct* and *complete*: a correct explanation covers at least the most important concepts and focuses on the behavior of the line, and a complete explanation covers all and only the concepts expected for that line.

### 3.2 LLM Assessment of Explanations

ESSE scores each submitted explanation on two dimensions: *correctness*, a binary judgment of whether the explanation captures the line's behavior, and *completeness*, a continuous score in $[0, 1]$, with an explanation deemed "sufficiently complete" at $\geq 0.5$. A single prompt to GPT-4o mini performs the assessment: given the problem statement, the worked example, the target line, and an expert explanation as context, the model returns the correctness label, the completeness score, and the numbers of expected concepts that are present in and absent from the student's explanation (Figure 2). The expert explanation is supplied only as context for judgment, not as an answer key to match, so the assessment requires no exact reference answer at scoring time.

### 3.3 Layered Feedback

Feedback is delivered as four levels of increasing scaffolding, unlocked progressively: the student reaches the next level only after engaging the current one and revising their explanation.

- **Flags** - shown after every submission: a correctness indicator (green/red, labelled correct or incorrect) and a completeness indicator (a bar with a green, yellow "sufficiently complete," or red state).
- **Samples** - peer-written explanations from a prior dataset [6] that a student can browse for extra detail, including examples attributed to more and less experienced students.

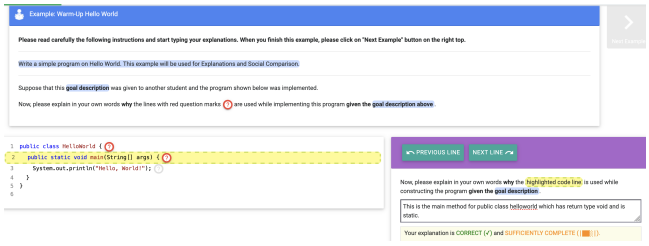


**Figure 1: The ESSE self-explanation tutor. Students explain a selected line of a worked example in their own words and receive immediate LLM feedback on the *correctness* and *completeness* of the explanation.**

```
# Task
Assess the **STUDENT EXPLANATION** for the specified **LINE NUMBER** in the given **SOURCE CODE**
based on accuracy, completeness, expected concepts, missing concepts, and included concepts.
## Evaluation Criteria
### 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**.
### Completeness
- An explanation is incomplete if it lacks details present in the **EXPERT EXPLANATION** that are necessary
for a full understanding of the given line of code.
### Expected Concepts
- The key concepts that a correct explanation should include, either based on the **EXPERT EXPLANATION**
or essential for explaining the line of code.
### Missing Concepts
- Expected concepts that are absent from the **STUDENT EXPLANATION**, reducing its completeness.
- These must be a subset of the Expected Concepts.
### Present Concepts
- Expected concepts that are correctly included in the **STUDENT EXPLANATION**, contributing to correct-
ness and completeness.
- These must be a subset of the Expected Concepts.
## Evaluation Guidelines
- **Correctness**: Assign '1' if the explanation is correct, otherwise '0'.
- **Completeness**: Compute as '(Number of Concepts Present) / (Number of Expected Concepts)', yielding a
floating-point value between '0' and '1' (e.g., '0.21354').
- **Reasoning**:
- Provide a justification for correctness by referencing relevant code details.
- Address the student directly using "you".
- Clearly separate **Correctness** from **Reasoning**.
- **Output Format**: Return the response in JSON format.
-
## Provided Information
**Program Description:**
{program_desc}
**Source Code:**
{source_code}
**Line Number:**
{line_num}
**Code:**
{line_content}
**Student Explanation:**
{student_explanation}
**Expert Explanation:**
{expert_explanation}
-
## Evaluation Output
{{ "Correctness": 1 or 0, "Completeness": any float value between 0 and 1, "Expected Concepts": ["EC1", "EC2",
...], "Concepts Missing": ["C1", "C2", ...], "Concepts Present": ["C3", "C4", ...], "CorrectnessReasoning": "[Provide a
clear justification for correctness, referencing code behavior and syntax.]", "CompletenessReasoning": "[Provide
a clear justification for completeness, referencing code behavior and syntax.]" }}
```

**Figure 2: The evaluation prompt used by ESSE. A single prompt produces all three quantities from the student explanation and its context.**

- **Feedback** - an LLM-generated message naming the concepts the explanation is missing and how to improve it, which the student can rate (like/dislike).
- **Answer** - an expert explanation of the line.

Flags appear immediately, but Samples, Feedback, and the Answer are gated: the Feedback and Answer options stay locked until the student has consulted the earlier levels and resubmitted, so the tutor

**Table 1: The two datasets and the research questions they support.**

| Dataset | Source | Size | Used for |
|---|---|---|---|
| CS1 study | Deployment | $N$=8; 409 attempts | RQ1, RQ2 |
| Crowd | MTurk | 124 raters; 1,696 ratings | RQ1 |

withholds the solution until students have attempted revision themselves. Advancing to the next line requires a correct explanation; completeness is instead encouraged through the flags.

## 4 Study Design and Data

We draw on two datasets (Table 1): a deployed CS1 pilot and an independent crowd re-rating of its explanations.

*Deployed CS1 study (RQ1, RQ2).* As a pilot of ESSE, we deployed it as a recitation activity in an introductory Java course (CS1). All $N = 8$ enrolled students participated under consent. Following the design of prior studies in this line of work [15], students first took a pretest assessing prior Java knowledge; they then worked through ESSE, explaining 30 lines of code across four intermediate worked examples (conditions, loops, objects); and finally took an isomorphic posttest. Participation earned extra credit and was not graded on correctness, to encourage genuine attempts rather than answer-matching. ESSE logged every interaction: students produced 409 attempts, of which 407 received feedback. At the end of the session, students completed a short survey on their perceptions of the tool and its feedback. A research expert annotated each live explanation–attempt for correctness and completeness, providing the human standard for RQ1. The change and agreement analyses below use the 358 feedback events that pair an attempt with its predecessor.

*Independent crowd study (RQ1).* To obtain a second, independent set of human labels, we recruited a crowd on Amazon Mechanical Turk to re-rate live-study explanations: 124 workers produced 1,696 ratings over 216 explanations on the same correctness and completeness rubric, with uneven coverage across explanations. Aggregated non-expert judgments can approximate expert labels for many annotation tasks [38]. We filter unreliable workers by chance-corrected agreement against the majority ($\kappa$-vs-majority), since raw agreement passes "base-rate clickers" who always select the majority label [16, 31], and aggregate the reliable raters into a single "community-wisdom" label per explanation.

*Analysis and notation.* For behavioral models (RQ2) we fit mixed-effects models with a random intercept for student (`user_id`) and report Satterthwaite-approximated significance. For agreement (RQ1) we report precision, recall, and F1 with McNemar's test for the binary correctness dimension, and point-biserial correlation and AUC for the continuous-versus-binary completeness comparison; crowd worker reliability is scored by chance-corrected agreement against the majority ($\kappa$-vs-majority). Throughout, we denote significance as $p < .05$ ($*$), $p < .01$ ($**$), and $p < .001$ ($***$), and follow APA convention in dropping the leading zero from coefficients bounded by one.

## 5 Evaluation

### 5.1 RQ1: Building an Interface with Reliable Automated Feedback

The interface needs a feedback engine whose judgments students can act on, so we ask how well the LLM's correctness and completeness scores agree with human judgment. We have two independent sets of human labels-a single expert (§5.1.1) and a crowd of non-expert raters (§5.1.2)-and we report how well the LLM does against each as a standard in turn.

#### 5.1.1 *LLM vs. the Expert.*

*Correctness.* On the deployed study's 358 rated attempts, treating the expert's "correct" label as the positive class, the LLM reaches precision .95 and recall .87 (F1 = .91; accuracy 83.8%). The gap between precision and recall is the telling part: the LLM misses correct explanations far more often than it accepts incorrect ones-42 false negatives against 16 false positives (McNemar $p = .001$), so it systematically *under-scores* correctness. Chance-corrected agreement is correspondingly only fair ($\kappa = .32$) under the skewed label distribution.

*Completeness.* Because the LLM emits completeness as a continuous score while the expert records a binary judgment, we treat the expert label as the outcome and ask how well the score separates it: the LLM's completeness score is a significant predictor (point-biserial $r = .39$, $p < .001$; AUC = .67) but poorly calibrated in absolute terms (mean absolute error = .45 against the binary label), tracking human judgment while falling short of a clean separation.

#### 5.1.2 *LLM vs. the Crowd.*

The crowd gives a second, independent human standard. Because each explanation was rated by several workers, we aggregate their ratings into one "community-wisdom" label per explanation and dimension. Raw agreement would let base-rate "clickers"-who select the majority label on nearly every item under the skewed distribution-count as reliable, so we score each worker by chance-corrected agreement with the crowd majority ($\kappa$-vs-majority), keeping 58 reliable raters (9 unreliable; the rest rated too few items to assess), and form each crowd label from a reliability-weighted majority vote.

*Correctness.* Against the crowd standard (216 explanations), the LLM reaches precision .90 and recall .95 (F1 = .93). Its few errors lean toward false positives (19) over false negatives (9)-the reverse of the expert comparison, where the LLM withheld credit; against the more lenient crowd it grants it.

*Completeness.* Completeness agreement is weaker, consistent with the crowd itself agreeing less on this dimension. The LLM's continuous completeness score orders the crowd judgment moderately (AUC = .76) but is poorly calibrated in absolute terms (mean absolute error = .42 against the crowd's fraction-complete).

### 5.2 RQ2: What Students Engage With, and What Impacts Learning

#### 5.2.1 *Intrinsic: Change in Explanation Quality.*

The impact question for RQ2 is whether the feedback moves students toward better explanations. A good explanation covers the concepts a line of code

**Table 2: Share of attempts followed by another attempt ("revise"), by the correctness and completeness of the current attempt. Students revise more when the feedback flags a problem.**

| Current attempt | Attempts | Revise |
|---|---|---|
| *by correctness* | | |
| Incorrect | 62 | 63% |
| Correct | 296 | 35% |
| *by completeness* | | |
| ≤ 25% | 14 | 64% |
| 25–50% | 271 | 42% |
| 50–75% | 66 | 30% |
| > 75% | 7 | 0% |

requires, so we ask whether students revise in response to feedback and whether their explanations then become more *complete*-by adding the *right concepts* rather than simply more words.

*Students revise when the feedback flags a problem.* Students made 1.66 attempts per explanation on average (median 1, max 7), and revision does not taper off with effort: 38% of explanations receive a second attempt (83 of 216) and 37% of those a third (31 of 83), with the per-step continuation rate holding or rising among the smaller group who go further. Whether they attempt again is strongly conditioned on the feedback they receive (Table 2): they revise 63% of the time after an *incorrect* verdict but only 35% after a correct one, and their willingness to revise climbs steadily as completeness falls—from 0% when the explanation is already near-complete to about two-thirds when it is least complete. Students treat the feedback as actionable, revising precisely when it signals a gap and stopping when the explanation is good.

*Revised explanations become more complete, by adding concepts.* Across successive attempts, completeness scores rise substantially (Wilcoxon $r = .84$, $p < .001$). The gain reflects *what* students add, not merely how much: completeness is *negatively* correlated with the change in word count (Spearman $r = -.49$, $p < .001$), and in a mixed-effects model predicting completeness from the number of distinct concepts and the word count (random intercept per student), conceptual content dominates volume ($\beta_{\text{concept}} = .50^{***}$ vs. $\beta_{\text{volume}} = .33^{***}$). Because completeness is defined by concept coverage, students improve by supplying the concepts a line requires rather than padding their prose-the behavior the feedback is meant to induce.

#### 5.2.2 *Extrinsic: Learning Gain from Pretest to Posttest.*

*Learning gain by prior knowledge.* Only 3 of the 8 students completed the posttest; the other 5 did not and are excluded rather than scored zero. Among the three completers, normalized gains were .67, −.25, and .78, and the highest gain came from the student with the lowest pretest (1 of 9). The pilot is too small for this to be more than suggestive.

#### 5.2.3 *Student Perceptions.*

The survey corroborates the behavioral findings (Figure 3, Table 3). First, students rated the feedback well overall: six of eight disagreed that it was irrelevant (1e) or incorrect or misleading (1f), and six of eight disagreed that they would use the system *without* feedback (1h)-they valued it. Two items line up directly with the objective results above: five of eight agreed the feedback helped them write more (1c), matching the measured growth in explanation length, and seven of eight agreed it helped them recall the concepts to include (1d), matching the concept-driven completeness gains. Students were also clear on the activity, understanding what a correct explanation should be (1a) and the goal of the task (1i); the least decisive responses concerned the definition of a *complete* explanation (1b) and knowing what to do next from a given feedback message (1k).

**Table 3: Survey responses ($N = 8$): counts agreeing (A; strongly agree + agree), neutral (N), and disagreeing (D; disagree + strongly disagree). †Items phrased so that *disagreement* is the favorable response.**

| | Statement (abbreviated) | A | N | D |
|---|---|---|---|---|
| 1a | Understood "correct" definition | 7 | 1 | 0 |
| 1b† | Did not understand "complete" definition | 1 | 2 | 5 |
| 1c | Feedback helped write longer explanations | 5 | 1 | 2 |
| 1d | Feedback helped recall key concepts | 7 | 1 | 0 |
| 1e† | Feedback was irrelevant | 1 | 1 | 6 |
| 1f† | Feedback was incorrect or misleading | 0 | 2 | 6 |
| 1h† | Would use the system without feedback | 0 | 2 | 6 |
| 1i | Feedback helped understand the task goal | 6 | 2 | 0 |
| 1k† | Unclear what to do next from feedback | 1 | 3 | 4 |

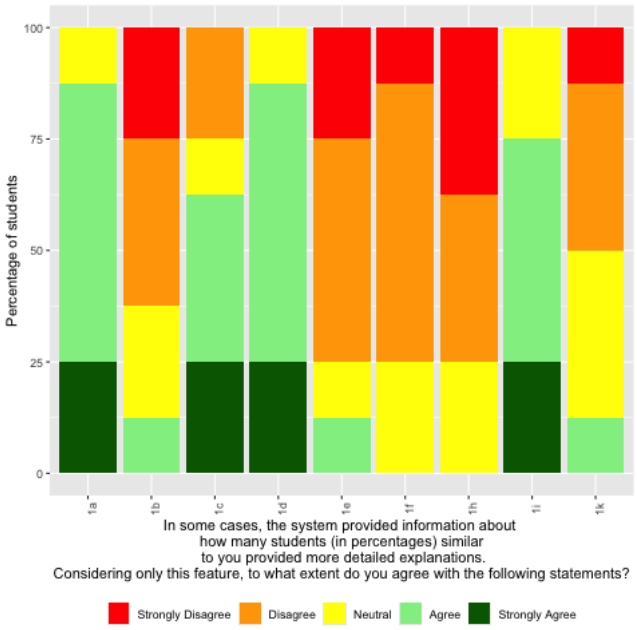


**Figure 3: Students' attitudes toward using the ESSE system.**

## 6 Discussion

*Correctness is reliable; its residual error is standard-dependent.* The LLM's correctness judgments agree strongly with both human standards (F1 = .91 against the expert, .93 against the crowd), which is

what a tutor needs to act on them. The small residual disagreement points in opposite directions depending on the standard: against the stricter expert the LLM *withholds* credit (more false negatives), while against the more lenient crowd it *grants* it (more false positives), so no single bias dominates. In a formative self-explanation setting the under-scoring direction is the safer one-an unearned "incorrect" nudges a student to revise, and we observe that students do revise rather than quit (RQ2), whereas an unearned "correct" would let a misconception stand.

*Choose metrics that fit skewed labels and two-part judgments.* Our RQ1 analysis is a methodological caution. With the skewed label distributions typical of "mostly correct" student work, a single agreement number-raw agreement or Cohen's $\kappa$-is misleading; precision and recall separate the kinds of error that matter for feedback. For a continuous completeness score, ordering (AUC) and absolute calibration (MAE) can diverge sharply-ours ranks completeness moderately well yet is poorly calibrated-so both should be reported rather than a single figure. Filtering raters by chance-corrected reliability before aggregating a crowd standard is a further safeguard against base-rate noise.

*Make completeness, and its concepts, the actionable target.* Since completeness-driven by conceptual content-is what tracks revision and improvement, feedback should foreground *which concepts are missing* rather than reward length. Our concept-level feedback is one realization of this; the result argues for designing feedback around concepts in general.

*Design implications for self-explanation tools.* As a pilot, the study also yields concrete design guidance. The faded scaffolding-withholding the expert answer until later levels-coexisted with productive persistence: students revised rather than quit (RQ2), suggesting the fade from flags to samples to feedback to answer is worth retaining rather than short-circuiting with an early reveal. And because conceptual completeness is the signal that moves learning, the interface should surface missing concepts as the primary feedback message rather than emphasizing a raw score or word count.

## 7 Limitations

The deployment is a single CS1 section with $N = 8$ students-a pilot-so RQ2 results, especially the learning-gain pattern, are suggestive rather than confirmatory; only 3 of 8 students have valid posttest data (a posttest of 0 indicates non-completion), and the perception-survey data is preliminary and small. The human standard in RQ1 is one expert; we add a second, independent crowd standard (also RQ1) but cannot rule out a shared blind spot. We also evaluate a single LLM grader under a single prompt configuration.

## 8 Future Work

This pilot opens several directions. First, we plan to move from pilot to confirmatory evidence with a larger, semester-long study across multiple CS1 sections, powered for the pretest–posttest learning-gain analysis that $N = 8$ could only suggest. Second, ESSE is designed as smart content for a personalized practice portal, and a natural next step is to integrate it into Mastery Grids [23] so that the concepts ESSE detects as *present*, *missing*, and *expected* feed the same concept-level open learner model that drives the portal's other activities-letting self-explanation evidence update the knowledge model alongside problem solving. Third, those concept-level signals could in turn drive adaptive support: recommending the next worked example, or surfacing targeted feedback, based on the concepts a learner repeatedly omits-an approach our group has found especially valuable for lower-prior-knowledge students [2]. Fourth, to remove the single-rater coverage gap, we are running a top-up crowdsourcing round on the 72 under-covered explanations (96 additional ratings) to reach at least three independent crowd raters per explanation; this lets us exclude the researcher from the crowd entirely and treat the expert labels as a clean held-out ground truth rather than a participant in the consensus. Finally, on the evaluation side, we will address the grader's under-scoring bias through calibration and prompt refinement, compare alternative (including open-weight) LLM judges, measure the same-prompt reliability of the judge, and collect graded rather than binary human completeness to enable a richer continuous-to-continuous validation.

## 9 Conclusion

We deployed an LLM-powered self-explanation tutor in CS1 and asked whether its automated feedback is trustworthy enough to drive the tutor and whether it helps students. On live data the LLM's correctness judgments agree strongly with two independent human standards-a single expert and a reliability-filtered crowd-though completeness remains the harder dimension to score. And the feedback changes behavior in the right way: students revise when it flags a problem and stop when their explanation is good, and their explanations grow more complete by adding the concepts a line requires rather than more words. Together these results give cautiously positive evidence for LLM-assisted self-explanation in introductory programming, while making its evaluative blind spots explicit.

## Acknowledgments

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

[1] 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*. 182–187.

[2] Jordan Barria-Pineda, Kamil Akhuseyinoglu, Stefan Želem-Ćelap, Peter Brusilovsky, Aleksandra Klasnja Milicevic, and Mirjana Ivanovic. 2021. Explainable recommendations in a personalized programming practice system. In *International conference on artificial intelligence in education*. Springer, 64–76.

[3] Katerine Bielaczyc, Peter L. Pirolli, and Ann L. Brown. 1995. Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem Solving. *Cognition and Instruction* 13, 2 (1995), 221–252.

[4] 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*. 337–340.

[5] Maia Caughey and Kasia Muldner. 2023. Investigating the utility of self-explanation through translation activities with a code-tracing tutor. In *International Conference on Artificial Intelligence in Education*. Springer, 66–77.

[6] 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 (May 2025). doi:10.32473/flairs.38.1.138727

[7] Jeevan Chapagain and Lasang Tamang. 2022. Automated assessment of student self-explanation during source code comprehension. In *The international FLAIRS conference proceedings*, Vol. 35.

[8] Michelene TH Chi. 2018. Learning from examples via self-explanations. In *Knowing, learning, and instruction*. Routledge, 251–282.

[9] Michelene TH Chi and Ruth Wylie. 2014. The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist* 49, 4 (2014), 219–243.

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

[11] Paul Denny, David H Smith IV, 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. *arXiv preprint arXiv:2403.06050* (2024).

[12] Geela Venise Firmalo Fabic, Antonija Mitrovic, and Kourosh Neshatian. 2019. Evaluation of Parsons Problems with Menu-Based Self-Explanation Prompts in a Mobile Python Tutor. *International Journal of Artificial Intelligence in Education* 29, 4 (Dec. 2019), 507–535.

[13] Arthur C. Graesser, Peter Wiemer-Hastings, Katja Wiemer-Hastings, Derek Harter, Tutoring Research Group Tutoring Research Group, and Natalie Person. 2000. Using Latent Semantic Analysis to Evaluate the Contributions of Students in AutoTutor. *Interactive Learning Environments* 8, 2 (2000), 129–147. arXiv:https://doi.org/10.1076/1049-4820(200008)8:2;1-B;FT129 doi:10.1076/1049-4820(200008)8:2;1-B;FT129

[14] Roya Hosseini, Kamil Akhuseyinoglu, Peter Brusilovsky, Lauri Malmi, Kerttu Pollari-Malmi, Christian Schunn, and Teemu Sirkiä. 2020. Improving engagement in program construction examples for learning Python programming. *International Journal of Artificial Intelligence in Education* 30, 2 (2020), 299–336.

[15] Roya Hosseini, Kamil Akhuseyinoglu, Andrew Petersen, Christian D Schunn, and Peter Brusilovsky. 2018. PCEX: interactive program construction examples for learning programming. In *Proceedings of the 18th Koli calling international conference on computing education research*. 1–9.

[16] Dirk Hovy, Taylor Berg-Kirkpatrick, Ashish Vaswani, and Eduard Hovy. 2013. Learning whom to trust with MACE. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1120–1130.

[17] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing code explanations created by students and large language models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 124–130.

[18] Arun-Balajiee Lekshmi-Narayanan and Peter Brusilovsky. 2024. Evaluating Correctness of Student Code Explanations: Challenges and Solutions. In *8th Educational Data Mining in Computer Science Education (CSEDM) Workshop at EDM2024 (CEUR Workshop Proceedings, Vol. 3796)*, Yang Shi, Peter Brusilovsky, Bita Akram, Thomas Price, Juho Leinonen, Ken Koedinger, and Andrew Lan (Eds.). CEUR.

[19] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. 2025. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 2757–2791.

[20] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[21] Marcia Linn. 1992. Can experts' explanations help students develop program design skills. *International Journal on the Man-Machine Studies* 36 (1992), 511–551.

[22] Siyang Liu, Xiaorong Guo, Xiangen Hu, and Xin Zhao. 2024. Advancing generative intelligent tutoring systems with GPT-4: design, evaluation, and a modular framework for future learning platforms. *Electronics* 13, 24 (2024), 4876.

[23] Tomasz Dominik Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. 2014. Mastery grids: An open-source social educational progress visualization. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. 357–357.

[24] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research*. 101–112.

[25] Priti Oli, Rabin Banjade, Jeevan Chapagain, and Vasile Rus. 2024. Automated Assessment of Students' Code Comprehension using LLM. In *Proceedings of the 2024 AAAI Conference on Artificial Intelligence (Proceedings of Machine Learning Research, Vol. 257)*, Muktha Ananda, Debshila Basu Malick, Jill Burstein, Lydia T. Liu, Zitao Liu, James Sharpnack, Zichao Wang, and Serena Wang (Eds.). PMLR, 118–128. https://proceedings.mlr.press/v257/oli24a.html

[26] Priti Oli, Rabin Banjade, Jeevan Chapagain, and Vasile Rus. 2024. Automated Assessment of Students' Code Comprehension using LLM. In *Proceedings of the 2024 AAAI Conference on Artificial Intelligence (Proceedings of Machine Learning Research, Vol. 257)*, Muktha Ananda, Debshila Basu Malick, Jill Burstein, Lydia T. Liu, Zitao Liu, James Sharpnack, Zichao Wang, and Serena Wang (Eds.). PMLR, 118–128. https://proceedings.mlr.press/v257/oli24a.html

[27] Priti Oli, Rabin Banjade, Arun Balajiee Lekshmi Narayanan, Peter Brusilovsky, and Vasile Rus. 2024. Exploring the effectiveness of reading vs. tutoring for enhancing code comprehension for novices. In *Proceedings of the 39th ACM/SIGAPP symposium on applied computing*. 38–47.

[28] Priti Oli, Rabin Banjade, Arun Balajiee Lekshmi Narayanan, Peter Brusilovsky, and Vasile Rus. 2023. When Is Reading More Effective than Tutoring? An Analysis through the Lens of Students' Self-Efficacy among Novices in Computer Science. *Grantee Submission* (2023).

[29] Priti Oli, Rabin Banjade, Andrew M Olney, and Vasile Rus. 2024. Can LLMs Identify Gaps and Misconceptions in Students' Code Explanations? *arXiv preprint arXiv:2501.10365* (2024).

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *ACL*.

[31] Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. 2010. Learning from crowds. *Journal of machine learning research* 11, 4 (2010).

[32] 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)*. 3982–3992.

[33] Vasile Rus, Kamil Akhuseyinoglu, Jeevan Chapagain, Lasang Tamang, and Peter Brusilovsky. 2021. Prompting for Free Self-Explanations Promotes Better Code Comprehension.. In *5th Educational Data Mining in Computer Science Education (CSEDM) Workshop at EDM2021*, Vol. 3051. CEUR. http://ceur-ws.org/Vol-3051/CSEDM_13.pdf

[34] Vasile Rus, Rajendra Banjade, Mihai Lintean, Nobal Niraula, and Dan Stefanescu. 2013. SEMILAR: A Semantic Similarity Toolkit for Assessing Students' Natural Language Inputs. In *Educational data mining 2013*.

[35] Vasile Rus, Sidney D'Mello, Xiangen Hu, and Arthur Graesser. 2013. Recent advances in conversational intelligent tutoring systems. *AI magazine* 34, 3 (2013), 42–54.

[36] Amir Shareghi Najar and Antonija Mitrovic. 2013. Examples and tutored problems: How can self-explanation make a difference to learning?. In *International Conference on Artificial Intelligence in Education*. Springer, 339–348.

[37] Naaz Sibia, Angela Zavaleta Bernuy, Elexandra Tran, Jessica Jia-Ni Xu, Joseph Jay Williams, Andrew Petersen, and Michael Liut. 2024. Exploring Self-Explanations in a Flipped Database Course. In *Proceedings of the 3rd International Workshop on Data Systems Education: Bridging Education Practice with Education Research* (Santiago, AA, Chile) *(DataEd '24)*. Association for Computing Machinery, New York, NY, USA, 20–26. doi:10.1145/3663649.3664374

[38] Rion Snow, Brendan O'connor, Dan Jurafsky, and Andrew Y Ng. 2008. Cheap and fast–but is it good? evaluating non-expert annotations for natural language tasks. In *Proceedings of the 2008 conference on empirical methods in natural language processing*. 254–263.

[39] Lasang Jimba Tamang, Zeyad Alshaikh, Nisrine Ait Khayi, Priti Oli, and Vasile Rus. 2021. A Comparative Study of Free Self-Explanations and Socratic Tutoring Explanations for Source Code Comprehension. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 219–225.

[40] Jessica Wen, Bianca Arteaga Alvarez, Jorge Moreno Velasco, Naaz Sibia, Angela Zavaleta Bernuy, Carlos Suarez Hernandez, Andrew Petersen, and Michael Liut. 2025. Self-Explanations: Does Timing Matter?. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 2* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 761. doi:10.1145/3724389.3730767

[41] Jessica Wen, Angela Zavaleta Bernuy, Naaz Sibia, Andrew Petersen, and Michael Liut. 2025. Enhancing Self-Explanation in Student Learning Through Large Language Models. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 2* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 762. doi:10.1145/3724389.3730790

[42] Fridolin Wild, Christina Stahl, Gerald Stermsek, and Gustaf Neumann. 2005. *Parameters driving effectiveness of automated essay scoring with LSA*. Technical Report. Loughborough University.

[43] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with BERT. *arXiv preprint arXiv:1904.09675* (2019).

Received ; revised ; accepted