CoTAL: Human-in-the-Loop Prompt Engineering, Chain-of-Thought Reasoning, and Active Learning for Generalizable Formative Assessment Scoring

Clayton Cohn*, Nicole Hutchins†, Ashwin T S*, and Gautam Biswas*
*Department of Computer Science, Vanderbilt University
Nashville, TN, USA
† College of Education, University of Florida
Gainesville, FL, USA
Email: clayton.a.cohn@vanderbilt.edu

***Abstract*—This document contains the experimental design details for applying CoTAL to each of the *Rules*, *Debugging*, and *Engineering Task* formative assessments discussed in the manuscript. The manuscript can be found here: https://arxiv.org/abs/2504.02323.**

RULES TASK

For the *Rules Task*, the training set consisted of 126 instances — four were used as few-shot examples in the prompt, and the remaining 122 served as the validation set. During Response Scoring, the human scorers' inter-rater reliability, as measured by Cohen's k, was 0.861, indicating "strong" agreement between scorers ($0.8 \leq \text{QWK} \leq 0.9$; [1]). The human scorers identified two *sticking points*. The first was that the scorers initially disagreed on whether or not to award a point for R3 ("less than set runoff") and R6 ("equal to set runoff") if a student set runoff to "none" instead of "0". The *Rules* correspond to specific code blocks in the SPICE environment, which is why one scorer felt that the student must explicitly set runoff to "0" (the runoff variable in the SPICE environment is a number, and there is no "none" option). However, after discussion, the two scorers agreed that students should receive credit for R3 and R6 if they set runoff to "none" because it is semantically equivalent to "0".

The second *sticking point* was whether or not to award students a point for R9 ("greater than set runoff") if they set runoff to "rainfall − absorption" instead of "rainfall − absorption limit." While absorption and absorption limit have the same value when rainfall is greater than the absorption limit, absorption and absorption limit represent distinct concepts in the SPICE environment (the former is related to the amount of rainfall and the absorption limit of the material the rain is falling on, whereas the latter is a constant value for a specific material). However, because absorption and absorption limit have the same value for the greater than condition, the scorers agreed to award students the point for R9 if they set runoff to "rainfall − absorption" instead of "rainfall − absorption limit." Additionally, we developed four guidelines during Response Scoring to help the LLM align with human consensus, encouraging it to "think" like human scorers. These guidelines, detailed in the *rules_task_prompt.txt* file in the Supplementary Materials, include instructing the LLM to cite the student's response when justifying scoring decisions and clarifying that the rules' ordering does not impact scoring.

During Prompt Development, we employed several prompt patterns that prior work demonstrated helps improve LLM performance [2]–[5]. First, we began with the *persona pattern* to instruct the LLM that it should play the role of a helpful teacher's assistant who helps teachers score middle school students' short answer formative assessment question responses. We utilized the *context manager pattern* to provide contextual information to the LLM to help guide its scoring and feedback. This included background information about the formative assessment, the formative assessment question, a correct "gold-label" (i.e., human-written) response, and the grading rubric. Additionally, we provided the LLM with the four guidelines that the human scorers agreed upon during Response Scoring. After the guidelines, we used the *template pattern* to request the LLM output its responses as JSON documents pursuant to a provided schema, which allowed for easier parsing of the LLM's generations.

The last step of *Rules Task* Prompt Development involved selecting the labeled few-shot examples. One instance, agreed on by the researchers during Response Scoring, was selected as a ground truth exemplar. This ensured the LLM was exposed to an unambiguous student response where all 3 rules (9 subscores) were correctly identified. Two instances were selected to address the two *sticking points* discussed at the beginning of this subsection (i.e., "none" being equivalent to "0" and 'rainfall − absorption" being equivalent to "rainfall − absorption limit" in the "greater than" condition). A fourth instance ensured data balance, representing each subscore by one positive and one negative instance in the few-shot examples. CoT reasoning chains were added to each labeled example to guide the LLM on whether to award a point for each subscore. For *sticking point* instances, these chains also addressed raters' reasons for disagreement during inter-rater reliability (IRR), helping the LLM align with human consensus. Cohn et al. (2024) [2] showed that instances difficult for human raters can similarly challenge the LLM.

To perform Active Learning, we ran our prompt through GPT-4 using each instance from the validation set. We then

analyzed the results to identify the LLM's scoring trend. For the *Rules Task*, the LLM had a tendency to overscore (i.e., there were more false positives than false negatives) by a ratio of roughly 2:1. The LLM's worst performance was on the "set absorption" subscores in each of the three conditions (R2, R5, and R8) and the "set runoff" subscore in the "greater than" condition (R9). To mitigate these issues, we selected one incorrectly predicted validation instance to insert back into the prompt:

> If rainfall is equal to absorption limit, then, there is no runoff and all is absorbed.
>
> If rainfall is less than absorption limit, then, there is no runoff and all is absorbed.
>
> If rainfall is greater than absorption limit, then, absorbed is absorption limit and runoff is all that wasn't absorbed

With this instance, the LLM incorrectly awarded points for R2 and R5 (the "set absorption" subscores for the "less than" and "equal to" conditions). Both errors occurred because the student said "all is absorbed" instead of explicitly defining the absorption value. Although this statement is true, human scorers required the student to explicitly set absorption to rainfall (or absorption limit for the "equal to" condition) to receive credit, as this understanding is crucial for translating knowledge into computational code blocks in the SPICE environment. The LLM's error predicting the R9 runoff subscore was similar, awarding a point for "runoff is all that wasn't absorbed." Again, human scorers required explicit setting of the runoff to either rainfall − absorption or rainfall − absorption limit to award credit.

We selected this instance to include in the prompt for several reasons. First, there was a large difference between the human total score and the LLM total score (the LLM predicted a perfect score of 9, while the human only awarded the student 6 points). Second, all three incorrectly predicted subscores were false positives, which addressed the LLM's scoring trend. Third, the LLM struggled with setting absorption values (R2 and R5). Finally, the incorrectly predicted subscore involving runoff was R9, which the LLM struggled with during Active Learning. This instance addressed the LLM's tendency to overscore, particularly in absorption and runoff subscores. We added the following CoT reasoning chains to this instance and inserted it back into the prompt:

> **R2 (Less Than Set Absorption)**: The student says 'all is absorbed' inside the 'less than' condition. While this statement is true (i.e., all the rainfall is absorbed), he or she does not explicitly set absorption equal to rainfall in the 'less than' condition per the rubric's guidance. Based on the rubric, the student earned a score of 0.
>
> **R5 (Equal To Set Absorption)**: The student says 'all is absorbed' inside the 'equal to' condition. While this statement is true (i.e., all the rainfall is absorbed), he or she does not explicitly set absorption equal to either rainfall or absorption limit in the 'equal to' condition per the rubric's guidance. Based on the rubric, the student earned a score of 0.
>
> **R9 (Greater Than Set Runoff)**: The student says 'runoff is all that wasn't absorbed' inside the 'greater than' condition. While this statement is true (i.e., all the rainfall that was not absorbed becomes runoff), he or she does not explicitly set runoff equal to either rainfall − absorption limit or rainfall − absorption in the 'greater than' condition per the rubric's guidance. Based on the rubric, the student earned a score of 0.

The final prompt for the *Rules Task* contained five few-shot examples (four were selected during Prompt Development, and one was added during Active Learning). Our full *Rules Task* prompt (with few-shot instances) is provided in the *rules_task_prompt.txt* file included in the Supplementary Materials.

DEBUGGING TASK

For the *Debugging Task*, the training set included 133 instances—six used as few-shot examples in the prompt, and the remaining 127 as the validation set. During response scoring, Cohen's κ was 0.740, indicating "moderate" agreement between scorers ($0.60 \leq QWK \leq 0.79$; [1]). Human scorers identified two *sticking points*. The first was whether to award a point if the specific blocks identified by the student were not unambiguously identifiable in the computational model. Scorers decided the exact block must be identifiable to assign a point. Students could identify blocks via line numbers, block colors, and block positions relative to other code (e.g., the absorption block in the first "if" statement). For example, if a student said, "the absorption block is wrong," they did not receive credit due to there being multiple absorption blocks. However, specifying "the absorption block *in the first 'if' statement* is wrong" was awarded a point for D2. The exception was D5, where mentioning that the absorption and absorption limit should be "swapped" was sufficient for credit, as there was only one place in the code needing this swap.

The second *sticking point* was whether to award students points for conflating the absorption and absorption limit blocks, which was similar to the *Rules Task*. While this was allowed in the *Rules Task*, the scorers decided this was unacceptable in the *Debugging Task*. In the *Rules Task*, scorers focused on the values set within each conditional statement, allowing interchangeable use of absorption and absorption limit for setting runoff in the "greater than" condition. However, in the *Debugging Task*, absorption and absorption limit are distinct blocks, so conflating them causes errors in the computational model. Thus, students would not receive credit for identifying absorption limit errors if they merely said "absorption" (and vice versa).

Similar to the *Rules Task*, we used Response Scoring to develop six LLM guidelines, agreed upon by the human scorers during the IRR process, to ensure the LLM aligned

with our researchers' scoring. Some guidelines, such as citing student responses as evidence for justifying scoring decisions, were consistent with those in the *Rules Task*. Others were specific to the *Debugging Task*, including prohibiting students from conflating absorption and absorption limit and requiring the unambiguous identification of code blocks to receive credit for identifying errors. The *Debugging Task* guidelines are provided in the *debug_task_prompt.txt* file in the Supplementary Materials.

During Prompt Development for the *Debugging Task*, we began with the *persona* and *context manager* patterns, instructing the LLM to be a helpful teacher's assistant for scoring formative assessments. We provided needed context, including background information, the task, block-based code errors (in textual form), an explanation of the code, the *gold-label response*, and the scoring rubric. We then added the six guidelines from Response Scoring. To close the prompt, we used the *template pattern* to provide a JSON schema for the LLM's outputs.

Importantly, context from the *Rules Task* and its rubric was essential for the LLM to accurately score and interpret student responses for the *Debugging Task* because students often referenced "if statements" in the code using terminology from the *Rules Task*. For example, a student wrote, "*in the third rule, it should be rainfall instead of absorption.*" If this context is omitted from the prompt, the LLM is likely to struggle in linking the student's response to the correct portions of the code and, by extension, the rubric.

The major difference between the *Debugging Task* prompt and the *Rules* and *Engineering Task* prompts, aside from domain and curricular context, is the use of the *meta language creation* pattern to help the LLM understand an "alternate language" [6]. In our case, the alternate language refers to the block-based code students used in the SPICE environment, which had to be provided and explained to the LLM.

For the *Debugging Task*, students saw a pictorial representation of a fictional student's computational model and were asked to debug it by identifying five errors. Students often used identifiers like line number and block color to refer to specific errors and blocks. Since we could not include images in our API calls to GPT-4, all context was provided in textual form. This raises an interesting question: *Can an LLM understand block-based code in textual form*? The SPICE environment uses a domain-specific modeling language (DSML [7]) built on NetsBlox[1], which is not well-represented in GPT-4's training data. The textual version of the code blocks used in the *Debugging Task* prompt is depicted and explained in Figure 1.

To complete the Prompt Development stage, we selected six instances to include as few-shot examples in the initial prompt for the *Debugging Task* (four to address ground truth and *sticking point* instances, and two to achieve data balance). One instance was chosen as a ground truth exemplar that was agreed upon by both scorers (the student correctly identified all five errors). Another instance addressed the issue of ambiguously identified code blocks, and a third addressed

[1]https://netsblox.org/

```
Here is the Fictitious Student's code:

when [Green Flag] clicked:          # (Line 1)
    set [Rainfall (inch)] to 1      # (Line 2)
    if [Rainfall (inch)] == [Absorption Limit (inch)]:          # (Line 3)
            set [Absorption (inch)] to [Rainfall (inch)]      # (Line 4)
            set [Runoff (inch)] to 0# (Line 5)
    set [Absorption Limit (inch) of the Selected Material]# (Line 6)
    if [Rainfall (inch)] < [Absorption (inch)]:         # (Line 7)
            set [Absorption (inch)] to [Absorption Limit (inch)]    # (Line 8)
            set [Runoff (inch)] to 0# (Line 9)
            if [Rainfall (inch)] > [Absorption Limit (inch)]: # (Line 10)
                    set [Absorption Limit (inch)] to [Absorption (inch)]    # (Line 11)
                    set [Runoff (inch)] to [Rainfall (inch)] - [Absorption Limit (inch)]      # (Line 12)
```

Fig. 1: SPICE environment block-based code distilled into raw text for GPT-4 to use for the *Debugging Task*. Tabs are used in Pythonic fashion, as prior work demonstrated GPT-4 understands Python syntax [8]. Items in brackets refer to objects such as variables or constants. Line numbers are provided as comments at the end of each line because students often reference line numbers in their formative assessment responses to identify specific pieces of code. Students may also refer to lines by color (lines 3, 7, 10, and 12 are green in the pictorial representation of the actual code depicted in Figure 2 in the manuscript). Students can also refer to "if" statements as "rules," as each "if" statement corresponds to an individual rule in the *Rules Task*. All of this information is provided to the LLM in the prompt under the textual code.

the conflation of the absorption and absorption limit blocks. A fourth instance highlighted using line numbers to identify individual blocks, which we hypothesized would help the LLM tie responses to relevant portions of the textual code for more accurate scoring. The last two instances ensured all five errors (i.e., subscores) were represented by both positive and negative examples.

During Active Learning, validation set instances were processed by the LLM using the constructed prompt. Our error analysis revealed that the LLM was equally likely to overscore as underscore (FP to FN ratio of 1:1). However, specific scoring trends emerged for individual subscores: (1) for D1 (set absorption limit before the first conditional statement), the LLM favored FPs to FNs by a ratio of 3:1; (2) for D4 (the greater than condition should not be nested in the less than condition), every error was a FP; and (3) for D5 (absorption and absorption limit should be swapped in the "greater than" condition), the LLM favored FNs to FPs by a ratio of 9:1. One validation set instance captured all three LLM errors, which we selected for inclusion in the prompt:

> They did not put in the absorption limit.
>
> absorption and absorption limit are flipped.
>
> the second if statement needs to be under the first.

In this instance, to address D1, the student incorrectly stated that the absorption limit was not set. It was set on line 6 of Figure 2 in the manuscript, but not at the beginning of the code block before the "less than" condition on line 3. Due to this nuanced distinction, the LLM incorrectly awarded a point for D1. For D4, the student incorrectly mentioned

that the second "if" statement should be under the first (the issue is the third "if" statement should not be nested inside the second). The LLM erroneously awarded a point for D4 due to this subtlety. For D5, the student correctly stated that absorption and absorption limit are flipped, but the LLM did not award a point. We hypothesized this was due to our guideline against awarding points for ambiguously defined blocks, despite scorers agreeing on an exception for this error since only one place in the code requires swapping absorption and absorption limit blocks. To address these issues, we added the following chains-of-thought to the validation instance:

> **D1 (Incorrectly Set Absorption Limit)**: The student says 'They did not put in the absorption limit'. This is incorrect. The code actually does set the absorption limit (incorrectly, on Line 6); however, the error is that it should have been set prior to the first 'if' statement (Line 3). Because the student does not identify that the absorption limit is initially set in the wrong part of the code or should be set before the first 'if' statement per the Incorrectly Set Absorption Limit rubric criteria, the student cannot be awarded a point for Incorrectly Set Absorption Limit. Based on the rubric, the student earned a score of 0.
>
> **D4 (Greater Than Nested in Less Than)**: The student says 'the second if statement needs to be under the first'. Saying that the second ('less than') conditional needs to be under the first ('equal to') conditional is incorrect. Actually, it is the third ('greater than') conditional that needs to be attached to, but not inside, the second ('less than') conditional. Because the student did not explicitly identify that the 'greater than' conditional statement is incorrectly nested inside the 'less than' conditional statement or that the 'greater than' conditional should be connected to the 'less than' conditional but not inside it pursuant to the Greater Than Nested in Less Than rubric criteria, the student does not earn a point for Greater Than Nested in Less Than. Based on the rubric, the student earned a score of 0.
>
> **D5 (Absorption Limit Swapped With Absorption)**: The student says 'the absorption and absorption limit are flipped.' While Rule 1 states that you need to know the exact piece of code the student is referring to based on the context the student provides in order to award the student a point, there is only one part of the code where the absorption and absorption limit blocks need to be switched to correct an error, so in this case we can assume the student is referring to the absorption and absorption limit blocks inside the 'greater than' condition. As such, the student correctly identifies that absorption and absorption limit should be swapped (i.e., absorption should be set to absorption limit) in the 'greater than' condition per the Absorption Limit Swapped With Absorption rubric criteria. Based on the rubric, the student earned a score of 1.

Unfortunately, we could not perform Active Learning for the *Debugging Task* due to GPT-4's token limitations. The *Debugging Task* required a longer prompt than the *Rules Task* because it included the computational model (in text form) and an explanation via the *meta language creation pattern*. Adding the instance caused the LLM to truncate its generations, exceeding GPT-4's 8,192 token context window. Thus, the final *Debugging Task* prompt contained only the six few-shot instances selected during Prompt Development. The final prompt (with few-shot instances) is in the Supplementary Materials in the *debug_task_prompt.txt* file. Future work will use LLMs like GPT-4-Turbo and GPT-4o, which have 128,000 token context windows.

ENGINEERING TASK

For the *Engineering Task*, the training set consisted of 129 instances — five were used as few-shot examples in the prompt, and the remaining 124 were used for validation. During Response Scoring, the human scorers' achieved inter-rater reliability of Cohen's $\kappa = 0.844$, indicating "strong" agreement between scorers [1]. The human scorers identified two *sticking points*. First, they debated what constitutes "fair tests." For example, should a student receive 3 or 4 points for stating the test is unfair because of unequal runoff or "too much" rain? Scorers agreed that to receive 4 points, the student must identify the different rainfall values as the reason for the unfair test, as rainfall is the independent variable. The second *sticking point* was whether a general understanding of design constraints qualified for 2 points if the student did not explicitly mention any constraint (e.g., cost, runoff amount, or accessible squares). Scorers agreed that students mentioning engineering constraint trade-offs would receive 2 points, even without naming specific constraints. However, this *sticking point* was not included in the initial prompt due to data balance issues, as adding another two-point instance risked biasing the LLM towards two-point responses, which represented only $\approx$15% of the dataset. Consequently, the LLM struggled with this *sticking point* on the validation set, which was corrected during Active Learning.

The human scorers established three guidelines during Response Scoring based on their discussion of key *sticking points*. The *Engineering Task* guidelines, presented in the *engineering_task_prompt.txt* file in the Supplementary Materials, include a general instruction to cite relevant portions of the student's response as evidence when justifying scoring decisions (as in previous tasks), along with task-specific guidelines like awarding the higher point value when students satisfy multiple scoring conditions in the rubric.

During Prompt Development, we used the *persona pattern* to inform the LLM of its role as a teacher's assistant for scoring formative assessments. The *context manager pattern* provided needed context, including background information, the formative question, and formatting information for student

responses. We included a human-written *gold-label response* for reference, illustrating what a correct response (i.e., 4 points) should look like. We also included the rubric and used the *template pattern* to provide the JSON schema for the LLM's response.

The final step of *Prompt Development* involved selecting and appending few-shot instances to the prompt. For the *Engineering Task*, we selected five instances: one for each possible rubric score (0 to 4) for data balance. Three instances (scores 0, 3, and 4) were ground truth exemplars, clearly defined in the rubric. The last two instances (scores 1 and 2) addressed the *sticking point* surrounding fair tests. One highlighted that not adding "the same runoff" to both models does not make the test unfair, and the other explained why "too much rain" is similarly not a valid reason. CoT reasoning chains were added to each labeled example to guide the LLM to align with the human scoring consensus for both ground truth and *sticking point* instances.

During Active Learning, we evaluated our prompt on the validation set. Our error analysis revealed that the LLM had a propensity for underscoring, with a FN-to-FP ratio of 2:1. This scoring trend was because the LLM awarded students 1 point when they should have received 2 points and 3 points instead of 4 points. The LLM's failure to score 2-point answers stemmed from the constraint satisfaction *sticking point*, initially unaddressed due to data imbalance concerns (i.e., students should receive 2 points for discussing constraint satisfaction trade-offs, even if not explicitly named). Additionally, students listing specific engineering constraints were still not awarded 2 points. The LLM underscored 4-point answers by relying on the words "fair" or "unfair" to award 4 points, missing instances with synonyms like "uneven," "inconsistent," "unequal," etc., which human scorers accepted. To address this, we added the following line to the rubric in the prompt to guide the LLM toward the human scoring consensus:

> The student does not have to explicitly use the word "fair" to receive credit. He or she can indicate that the tests are not fair by mentioning that the two tests are uneven, inconsistent, impossible to compare, etc.

To address the constraint satisfaction *sticking point*, we appended the following validation set instance to the prompt:

> **Answer**: No
> **Explanation**: You cannot determine which is better because you could have a different need and be okay with the deficit in absorption and cost when the accessible squares are higher like if you had more children who would need accessible squares.

We chose this instance specifically because the student mentioned individual design constraint considerations (e.g., cost and accessible squares) by name but was not awarded 2 points. We used the following CoT reasoning chain to align the LLM with the humans' consensus that 2 points should be awarded if: (1) the student demonstrates a general understanding of the engineering design constraints, or (2) if he or she mentions individual design constraints explicitly:

> The student provides an Answer of 'No', which demonstrates he or she understands the two designs cannot be compared. The student elaborates by saying 'You cannot determine which is better because you could have a different need', which indicates a general understanding of the trade-offs between the Engineering Constraints. Additionally, the student provides specific references to the Engineering Constraints by mentioning 'the deficit in absorption and cost when the accessible squares are higher'. This shows the student has a focused understanding of the Engineering Constraint considerations, as he or she mentions 'accessible squares' (Accessibility_Constraint), 'deficit in absorption' (Runoff_Constraint), and 'cost' (Cost_Constraint). While the student did not mention the different rainfall values and is therefore, ineligible to receive 3 or 4 points, both the student's general and focused understanding of the Engineering Constraints qualify him or her for 2 points, pursuant to the Rubric. Based on the Rubric, the student earned a score of 2.

At this point, the *Engineering Task* prompt was ready to be deployed for testing. Our complete *Engineering Task* prompt (including all few-shot instances) is provided in the *engineering_task_prompt.txt* file included in the Supplementary Materials.

REFERENCES

[1] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[2] C. Cohn, N. Hutchins, T. Le, and G. Biswas, "A chain-of-thought prompting approach with llms for evaluating students' formative assessment responses in science," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, pp. 23 182–23 190, Mar. 2024.

[3] C. Cohn, C. Snyder, J. Montenegro, and G. Biswas, "Towards a human-in-the-loop llm approach to collaborative discourse analysis," in *Artificial Intelligence in Education. Late Breaking Results*, A. M. Olney, I.-A. Chounta, Z. Liu, O. C. Santos, and I. I. Bittencourt, Eds. Cham: Springer Nature Switzerland, 2024, pp. 11–19.

[4] C. Snyder, N. Hutchins, C. Cohn, J. Fonteles, and G. Biswas, "Using collaborative interactivity metrics to analyze students' problem-solving behaviors during stem+c computational modeling tasks," *Accepted (pending minor revisions) at Learning and Individual Differences.*, 2025.

[5] C. Snyder, N. M. Hutchins, C. Cohn, J. H. Fonteles, and G. Biswas, "Analyzing students collaborative problem-solving behaviors in synergistic stem+c learning," in *Proceedings of the 14th Learning Analytics and Knowledge Conference*, 2024, p. N/A.

[6] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[7] N. M. Hutchins, G. Biswas, N. Zhang, C. Snyder, Á. Lédeczi, and M. Maróti, "Domain-specific modeling languages in computer-based learning environments: A systematic approach to support science learning through computational modeling," *International Journal of Artificial Intelligence in Education*, vol. 30, pp. 537–580, 2020.

[8] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. Tulio Ribeiro, and Y. Zhang, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," *arXiv e-prints*, p. arXiv:2303.12712, Mar. 2023.