MetaFeedback-AD: Metacognitive Feedback for Algorithm Design Problems Dataset

Abstract

Automated personalized feedback generation is crucial in scaling education to large and diverse student populations. While automated personalized feedback systems have been developed to address students' specific answers, they typically overlook individual metacognitive skills. However, educational experts emphasize that incorporating metacognitive information into feedback can significantly enhance students' self-regulated learning and long-term problem-solving abilities. The lack of datasets including student submissions and metacognitive profiles has hindered the development of such systems. This paper introduces MetaFeedback-AD, the first dataset to pair structured metacognitive self-assessments with student code submissions for algorithm design problems. It comprises two subsets: MetaFeedback-AD-Introductory, collected from first-year students and containing 16,803 instances across 71 problems and 516 unique metacognitive profiles; and MetaFeedback-AD-Competitive, collected during a hackathon involving more advanced students, featuring 372 instances across six complex problems. Each instance includes a problem description, a student's code, the correct solution, 16 metacognitive responses, a numerical metacognitive vector, and expert-aligned feedback. Further, MetaFeedback-AD supports two evaluation settings—Disjoint Metacognitive Profiles (DMP) and Shared Metacognitive Profiles (SMP)—to assess model generalization across learner types. This dataset enables the development of feedback generation systems that adapt to individual metacognitive characteristics.

CCS Concepts

• **Applied computing** → **Computer-assisted instruction**; **E-learning**; *Interactive learning environments.*

Keywords

Dataset, Metacognition, Personalized Feedback, Programming Education, Introductory level, Competitive level

ACM Reference Format:

. 2018. MetaFeedback-AD: Metacognitive Feedback for Algorithm Design Problems Dataset. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The demand for personalized learning support in education is growing, with increasing expectations for systems to tailor learning experiences to individual students' needs and skill levels. Automated feedback systems play a key role by delivering real-time, personalized feedback that enhances student learning while reducing educators' workload. Significant research and development have been conducted in language learning, math problem-solving, and programming [10, 12, 16]. Despite advancements in automated personalized system in programming education, they focus on providing students with corrected code solutions or helping them identify errors in their code [13] [4]. They fall short in integration with self-regulated learning strategies, which are crucial for fostering long-term problem-solving skills—an essential aspect of programming education[7] [14]. Further, recent studies highlight the importance of incorperating *metacognitive*

Recent studies have introduced a novel approach to enhancing students' learning experiences by focusing on developing metacognitive skills—a critical set of abilities closely aligned with self-regulated learning. These studies emphasize the importance of considering each student's metacognitive skills when designing learning environments [15], particularly in programming education. By addressing students' individual learning needs through metacognitive skill development, educational systems can foster deeper understanding, enhance problem-solving abilities, and promote long-term learning success. However, further research in this area is hindered by the lack of data containing students' metacognitive information.

To bridge this gap, we introduce a novel dataset that captures students' programming behaviors and metacognitive skills in algorithmic design problem-solving. Our contributions include:

- **A metacognition assessment questionnaire** designed to evaluate students' metacognitive strategies in algorithm design problem-solving.
- **A comprehensive dataset** comprising programming submissions from students across both introductory and competitive level tasks, enriched with metacognitive skill assessment data and expert-validated annotated metacognitive feedback for code answer submissions for introductory-level.

By providing a structured dataset with detailed metacognitive information, our work aims to facilitate research on adaptive learning systems, self-regulated learning strategies, and intelligent tutoring systems in programming education.

2 Metacognition Measuring Questionnaire

A metacognition measuring questionnaire is a structured tool designed to assess individuals' awareness and regulation of their own cognitive processes. It typically includes self-report items that evaluate aspects such as knowledge of cognition (e.g., declarative, procedural, and conditional knowledge) and regulation of cognition (e.g., planning, monitoring, and evaluation). These questionnaires

help researchers and educators understand how individuals engage in self-reflection, control their learning strategies, and adapt their thinking to solve problems effectively. But due to the inherent internal and complex nature, metacognition is difficulty to be observed directly. Various approaches have been proposed, including offline measures such as self-report questionnaires and interviews, as well as online measures like the Think Aloud Protocol and systematic observations by Veenman and van Cleef [19]. Although the Think Aloud Protocol provides valuable insights it is not a scalable option for large classes within Digital Learning Environments (DLEs) due to the labor-intensive and time-consuming nature of the process. Additionally, self-report questionnaires, which rely on retrospective memory, often exhibit low to moderate correlations with Think Aloud measures, with task-specific questionnaires showing more promising results [17].

Given the absence of a questionnaire specifically designed to assess metacognition in the programming domain, we have developed a new questionnaire by adapting and generalizing existing questionnaires from other domains. The adapted questionnaire is validated through expert evaluation to ensure its relevance and effectiveness within the programming context.

2.1 Developing the Questionnaire

We adapted the questionnaire proposed in Fernando and Wijeratne [3], which consists 16 questions under three categories: Planning, Monitoring and Evaluation, to measure the student's metacognition in mathematical problem-solving. While most questions in the questionnaire are generic, some were modified to improve clarity and better align with the programming domain. The adapted questionnaire, tailored to the programming problem-solving process, is shown in Table 1. Students are asked to rate how frequently they engage in each step of the problem-solving process by selecting one of three options: 1 (Almost Never), 2 (Sometimes), or 3 (Often) for each question. Rather than directly assessing metacognitive skills, the questions focus on observable problem-solving behaviors. This approach reduces the risk of students misjudging their metacognition skills by focusing on their problem-solving behaviors.

To ensure its relevance and robustness, the modified questionnaire was cross-validated against the PCDIT framework introduced by Kurniawan et al. [5]. This framework outlines a five-phase, non-linear approach comprising Problem Definition, Cases, Design of Algorithm, Implementation, and Testing, guiding novice programmers in transforming problem specifications into implemented and tested solutions.

2.2 Evaluating the Questionnaire

The modified questionnaire is validated using the method proposed by Taherdoost [18]. To assess the relevance of each question to the programming domain, four experts were asked to rate each item using a Likert scale. The scale includes the following ratings: 1 = the item is not relevant to the measured domain, 2 = the item is somewhat relevant to the measured domain, 3 = the item is quite relevant to the measured domain, and 4 = the item is highly relevant to the measured domain. This approach ensured that the modified questionnaire accurately reflects the key aspects of metacognition in programming problem-solving. The validation ratings provided by the experts are summarized in the Table 2. We assumed students' answers reflect honest self-evaluation of their behaviors as we are based on Think Aloud Protocol.

Table 1: Modified questionnaire for the programming domain

Planning	
PL1	I read the question entirely, before I start the solving process.
PL2	I identify and highlight the key requirements, inputs, outputs and constraints of the programming task.
PL3	I rephrase/summarize the question in my own words and identify the main points.
PL4	I create specific input examples and manually work through them to reach the outputs, to understand the problem better before thinking about the algorithm.
PL5	I break down the problem statement into smaller, achievable sub-goals before beginning the implementation
PL6	Before I solve the problem, I estimate/think about the nature of the possible algorithm by recognizing patterns such as repetition and conditional that I would get.
PL7	I sketch out the algorithm or plan the solution before start coding.
Monitoring	
MO1	Every time I revise and execute the designed algorithm systematically to reach the answer.
MO2	I am always vigilant on the implementation process to verify that I am on the correct way to the solution.
MO3	I pay attention to avoid negligent mistakes during the implementation process.
MO4	I keep an eye on the problem-solving steps which helps me to verify intermediate results.
MO5	I always monitor the ongoing program implementation process.
Evaluation	
EV1	I check if the algorithm is acceptable and compatible with given data constraints.
EV2	I confirm that the final implementation is correct.
EV3	I refer again to the problem statement and check if the implemented solution is acceptable for all the given problem requirements.
EV4	I refer to similar problems solved earlier and reflect on the accuracy and efficiency of my code solution.

Based on the expert validation provided and the discussions with experts, the final version of the questionnaire was constructed (Table 1). Minor modifications were made to the initial version of the modified questionnaire to improve the clarity of the questions. These modifications did not have an effect on the content of the questions. Hence, the expert validation performed for the initial version was considered valid for the final version. We performed a weighted percentage calculation to conclude the agreement of experts regarding the validity of the questionnaire.

$$AgreementPercentage = \frac{\sum_{i=1}^{4} ExpertRating}{\sum_{i=1}^{4} MaxRating(=4)} 100\% \quad (1)$$

As observed in Table 2, most experts agreed on the validity of the questionnaire. Hence, we proceeded to utilize it in the dataset creation process.

3 Introductory-Level Dataset

In this section, we introduce our dataset of introductory-level programming tasks, which includes real-world metacognitive profiles from undergraduate students. Each student solution is accompanied by annotated, expert-validated metacognitive feedback that aligns with both the state of the solution and the student's metacognitive skill level. The following subsections provide more details regarding

Table 2: Summary - Expert validation of metacognition measuring questionnaire

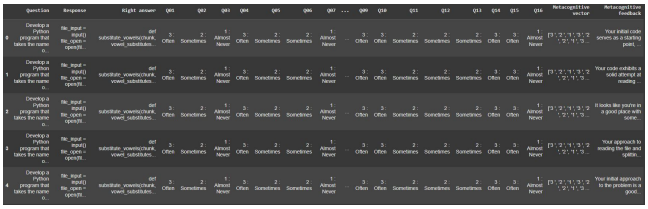
Q No	Expert 1	Expert 2	Expert 3	Expert 4	Agreement Percentage
1	4	4	4	4	100
2	4	4	4	4	100
3	4	2	2	2	62.5
4	4	2	3	3	75
5	2	4	3	4	81.25
6	2	3	3	4	75
7	4	2	4	2	75
8	4	2	3	3	75
9	4	3	3	3	81.25
10	4	1	4	3	75
11	4	3	4	4	93.75
12	4	1	3	3	68.75
13	4	3	4	4	93.75
14	4	1	4	4	81.25
15	4	2	4	3	81.25
16	2	4	3	3	75

the dataset, data collection, data processing, and expert validation of the annotated dataset.

3.1 Data Field Definition

The key data fields included in the final dataset are as follows:

- **Question:** An introductory-level programming problem. The dataset contains 71 different programming problems.
- **Response:** A code response from a student. The response can be in any of the start/ intermediate/ finish states. The response is in Python programming language.
- **Right answer:** The expected correct answer for the programming problem given in Python programming language.
- **Q01 - Q02 - ... - Q16:** 16 data fields each containing student's answer for each question in the metacognition measuring questionnaire. The answer is one out of three options; 1: Almost Never, 2: Sometimes, 3: Often.
- **Metacognitive vector:** The vectorized representation of the answers to the above 16 questions from the student. This is the student's metacognition profile. The final dataset contains 561 unique metacognitive vectors.
- **Metacognitive feedback:** Annotated metacognitive feedback that aligns with students' solution state and metacognition level.



	Question	Response	Right answer	Q01	Q02	Q03	Q04	Q05	Q06	Q07	...	Q09	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Metacognitive vector	Metacognitive Feedback
0	Develop a Python program that takes the name o...	file_input = input() file_open = open(fi...	def substitute_vowels(chunk, vowel_substitutes...	3 Often	2 Sometimes	1 Almost Never	3 Often	2 Sometimes	2 Sometimes	1 Almost Never	...	3 Often	3 Often	2 Sometimes	2 Sometimes	2 Sometimes	3 Often	3 Often	1 Almost Never	[3 2 1 3 3 2 2 1 1 3 ...	Your initial code serves as a starting point,...
1	Develop a Python program that takes the name o...	file_input = input() file_open = open(fi...	def substitute_vowels(chunk, vowel_substitutes...	3 Often	2 Sometimes	1 Almost Never	3 Often	2 Sometimes	2 Sometimes	1 Almost Never	...	3 Often	3 Often	2 Sometimes	2 Sometimes	2 Sometimes	3 Often	3 Often	1 Almost Never	[3 2 1 3 3 2 2 1 1 3 ...	Your code exhibits a solid attempt at reading ...
2	Develop a Python program that takes the name o...	file_input = input() file_open = open(fi...	def substitute_vowels(chunk, vowel_substitutes...	3 Often	2 Sometimes	1 Almost Never	3 Often	2 Sometimes	2 Sometimes	1 Almost Never	...	3 Often	3 Often	2 Sometimes	2 Sometimes	2 Sometimes	3 Often	3 Often	1 Almost Never	[3 2 1 3 3 2 2 1 1 3 ...	It looks like you're in a good place with some...
3	Develop a Python program that takes the name o...	file_input = input() file_open = open(fi...	def substitute_vowels(chunk, vowel_substitutes...	3 Often	2 Sometimes	1 Almost Never	3 Often	2 Sometimes	2 Sometimes	1 Almost Never	...	3 Often	3 Often	2 Sometimes	2 Sometimes	2 Sometimes	3 Often	3 Often	1 Almost Never	[3 2 1 3 3 2 2 1 1 3 ...	Your approach to reading the file and splitte...
4	Develop a Python program that takes the name o...	file_input = input() file_open = open(fi...	def substitute_vowels(chunk, vowel_substitutes...	3 Often	2 Sometimes	1 Almost Never	3 Often	2 Sometimes	2 Sometimes	1 Almost Never	...	3 Often	3 Often	2 Sometimes	2 Sometimes	2 Sometimes	3 Often	3 Often	1 Almost Never	[3 2 1 3 3 2 2 1 1 3 ...	Your initial approach to the problem is a good...

Figure 1: Sample of data records from the final introductory-level dataset

Figure 1 shows a sample set of data records presented in the final dataset.

3.2 Data Collection

The dataset is based on data collected from 1st year undergraduates. It includes programming problems and corresponding student answers collected during their first semester, specifically for the Programming Fundamentals module. The data was gathered from two assessments, PP1 and PP2, which were conducted under examination conditions in the Level 3 Laboratory at the Department . During these assessments, students had access to all module content available on the Moodle page and were permitted to bring their personal notebooks. However, internet access and any form of communication were strictly prohibited to maintain examination integrity. To capture students' metacognitive profiles, the questionnaire was administered separately to measure their metacognitive skill levels. Out of total 1,159 students, 581 participated by submitting responses to the questionnaire. Following data cleaning, the final dataset comprises 16,803 instances.

3.3 Data Annotation and Processing

3.3.1 Data Preprocessing. Initially, the data was collected as two separate datasets: one containing student responses to the metacognition questionnaire, and the other containing programming task data. These datasets were merged using student IDs, resulting in 21,890 instances. A data cleaning process was then applied, which included *Removing Null Values, Removing Duplicates, Removing Erroneous Values*, resulting in 16,803 instances.

3.3.2 Data Annotation. We created a vectorized representation of each student's responses to the 16 metacognition-related questions, resulting in a 16-dimensional vector referred to as the metacognitive vector or metacognitive profile. Metacognitive feedback was annotated for the preprocessed data instances. For each student submission, feedback was generated using GPT-4.o mini, taking into account both the student's code solution and their metacognitive profile. To ensure coherence between the feedback, the corresponding code, and the metacognitive information, we employed a chain-of-thought prompting strategy, introduced by Wei et al. [20]. The model was guided to: *1: Identify errors and potential optimizations in the student's code, 2: Interpret the student's metacognitive level based on their questionnaire responses and corresponding profile, 3:Recognize areas where the student demonstrates weaknesses in metacognitive skills, 4:Provide feedback that not only helps the student improve their code but also supports the development of their metacognitive abilities.*. The complete prompt used for feedback generation is included in Appendix A.

3.3.3 Data Postprocessing. To maintain the quality and integrity of the dataset, a post-processing step to clean the data was carried out to remove anomalies and outliers, particularly those introduced during the inclusion of the AI-generated "Metacognitive Feedback" field. The data cleaning process involved a comparative evaluation of metacognitive feedback generated by two language models: Mistral-large-latest (version 24.11) and GPT-4.o mini. The same chain-of-thought prompt in Appendix A was utilized for both models. 1)For each student submission, both models produced separate feedback entries, 2)A similarity score was computed between the two outputs, 3)The distribution of these similarity scores was then visualized to understand the overall alignment between the models,

and 4)Based on this distribution, a 95% confidence interval was calculated, and only the data points falling within a 5% error margin of this confidence interval were retained, ensuring consistency and reliability. The steps in the cleaning process is given in the figure 2.

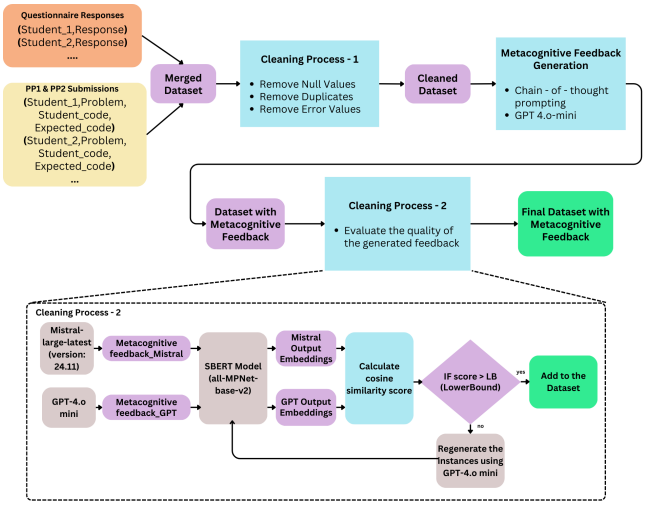


Figure 2: Data Annotation and Processing Pipeline

Various methods have been proposed to calculate similarity between natural language texts. Traditional metrics like BLEU [11] and ROUGE [6] focus on exact word or n-gram matches. Other approaches involve computing cosine similarity or Levenshtein distance on vector representations from models such as Word2Vec [8] or BERT [2], which capture semantic meaning. Recent studies on text deduplication [9, 21] suggest that using a pre-trained SBERT (Sentence Transformer) model [1] is more effective for semantic similarity. Therefore, we used SBERT and BERT Score to compare feedback generated by GPT-4.o mini and the Mistral-large model, and analyze the distribution of their cosine similarity scores.

Initially a significance level of 99.999% was achieved with a similarity score of 0.85711. The minimum similarity score observed was 0.0130, while the maximum was 0.9764, with a mean of 0.8587 and a standard deviation of 0.0005. Using the Inter Quartile Range (IQR) with a factor of 1.5, the calculated lower bound was 0.7050. The similarity score distribution plot is shown in figure 3. A total of 563 instances were identified with similarity scores below this lower bound. These 563 instances will be used to regenerate the metacognitive feedback using both models. The score distribution after the regeneration is shown in figure4. After analyzing the similarity scores, the minimum and maximum similarity scores were found to be 0.5059 and 0.9764, respectively. Using the initial lower bound ($q = 0.7050$), 563 instances were identified with similarity scores below this threshold. Following the regeneration of metacognitive feedback, the new lower bound increased to $q_{\text{after\_regeneration}} = 0.7136$. The number of instances with similarity scores below $q = 0.7050$ reduced significantly to 75, demonstrating improved alignment of feedback. We conducted further analysis over these 75 instances. The cumulative distribution of the similarity scores of these instances are plotted in the figure shown in 5.

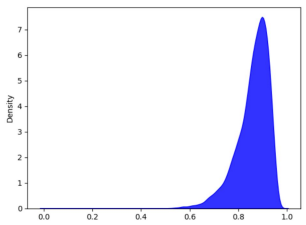


Figure 3: Similarity Score Distribution Before Data Cleaning

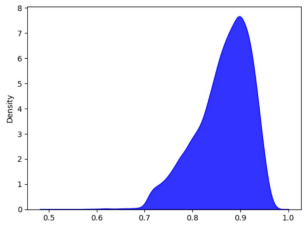


Figure 4: Similarity Score Distribution After Data Cleaning

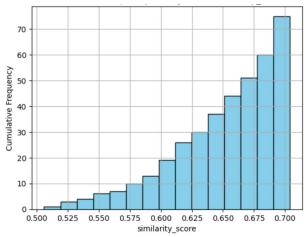


Figure 5: Similarity Score Distribution After Data Cleaning- Lower Bound

Analysis of the 75 instances with similarity scores below the initial lower bound ($q = 0.7050$) revealed that 39 instances have scores higher than 0.65. In these cases, GPT-generated feedback effectively identified code errors, provides specific corrections, and incorporates metacognitive strategies. However, Mistral-generated feedback in these instances tended to echo the metacognitive questionnaire and repeats its questions without addressing specific errors. For processing, the 39 instances with scores above 0.65 were considered valid, while the remaining 36 instances were manually reviewed and processed by regenerating.

3.4 Expert Validation

We adapted the validation criteria from the feedback evaluation criteria defined in the study Roest et al. [16]. As this study lacks the validation for the 'Metacognitive Alignment' of the feedback we included new criteria as *Metacognitive*. As some of the criteria mentioned in the research are not specific to our task and it causes some duplication in validation we dropped some criteria from the study. The criteria we have utilized in expert validation for the dataset are included in detail in Table 2.Each of the 30 instances is to be evaluated by three experts based on the five criteria mentioned above. The expert validation process is currently ongoing, and we have received responses from one expert so far.

The response from the expert is summarized in the Table 4. The expert's vote for each instance under each criterion is summed up in the tabel. The process of collecting expert validation responses is ongoing. Once this process is complete, statistical validation of the dataset can be performed.

Based on the current expert response, the annotated feedback appears to be of high quality in terms of feedback type. Most data instances include all four types of feedback, with every instance

Table 3: Expert Validation Criterias

Category	Options
Feedback Type	**Knowledge about Task Constraints (KTC):** Task-specific requirements such as required constructs or prohibited methods. **Knowledge about Concepts (KC):** Conceptual explanations or examples to aid understanding. **Knowledge about Mistakes (KM):** Identification and categorization of errors including test failures, compiler errors, style issues, etc. **Knowledge about How to Proceed (KH):** Guidance on how to correct mistakes or proceed with the next steps in the task.
Information	**Tip** or **Explanation** or **Compliment** or **Tip & Explanation** or **Tip & Compliment** or **Tip & Explanation & Compliment** or **No**
Personalized	**Yes** or **No** Indicates whether the feedback refers specifically to the student's code or approach.
Metacognitive	**Highly** or **Moderately** or **Lowly** Indicates whether the feedback is aligned with the student's metacognitive level.
Misleading Information	**Yes** or **No** Indicates whether the feedback contains any misleading or incorrect information.

containing both *Tips* and *Explanations*. Only six instances lack the *Compliment* component, indicating that the generated feedback is generally supportive from the student's perspective. Furthermore, all instances are *Personalized* to the student's approach and current solution state. In terms of *Metacognitive* alignment, the feedback ranges from moderate to high quality. The expert reviewer identified only six instances that may contain potentially *Misleading Information* that could confuse students. This indicates that while the overall quality is promising, further refinement could enhance the dataset even more.

3.5 Dataset Analysis

We performed data analysis over the dataset which is named as *full-dataset.csv*. The distribution of student responses to each metacognition question in the questionnaire is shown in Figure 6. Notably, students show a significantly lower frequency in sketching out an algorithm before starting to code (Q07), which corresponds to PL7 in Table 1. Additionally, students tend to lack practices such as summarizing a given question in their own words, creating input examples and manually working through them to understand the problem, and referring to previously encountered similar problems (Q03, Q04, Q16), corresponding to PL3, PL4, and EV4 in Table 1.

Table 4: Expert validation response summary

Category	Options	Expert vote
Feedback Type	KTC	27/30
	KC	28/30
	KM	26/30
	KH	30/30
Information	Tip	0/30
	Explanation	0/30
	Compliment	0/30
	Tip & Explanation	6/30
	Tip & Compliment	0/30
	Tip & Explanation & Compliment	24/30
	No	0/30
Personalized	Yes	30 /30
	No	0/30
Metacognitive	Highly	4/30
	Moderately	26/30
	Lowly	0/30
Misleading Information	Yes	6/30
	No	24/30

Almost all the students participated have a very high frequency for reading the question entirely and identifying key requirements (Q01, Q02), corresponding to PL1 and PL2 in Table 1.

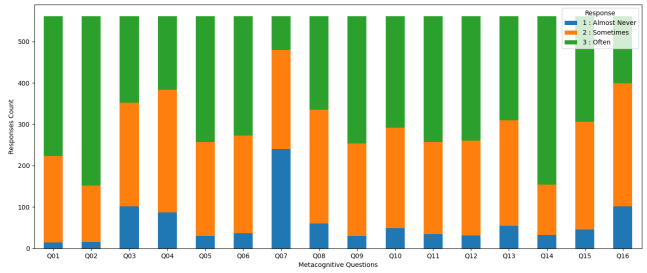


Figure 6: Distribution of students responses for metacognition measuring questionnaire - full dataset

Furthermore, we calculated an *Average Metacognition Score* by summing up all 16 values in the form of integer values(1: Almost Never, 2: Sometimes, 3:Often) and dividing them by 16 to average. The *Average Metacognition Score* distribution in the full dataset is illustrated in figure 6.

We introduce two types of train-validation-test splits for the above annotated and validated full dataset.

3.5.1 Distinct metacognitive profiles in train and test splits. The dataset split was designed with the following considerations: 1)Students included in the test set do not appear in the training set, 2)Students whose metacognitive profiles are similar to those in the test set were excluded from the training set and instead included in the validation set, 3)Programming problems related to the above excluded instances are tracked, and all the instances that are within the tracked programming problems are also excluded from the training set, 4)The excluded data instances are considered to be the validation dataset. As a result, the training and test sets are mutually exclusive not only in terms of student identities but also in terms of metacognitive profiles and programming problem exposure. The *train, validation, test* splits contains *12,371 , 889, 3543* instances respectively.

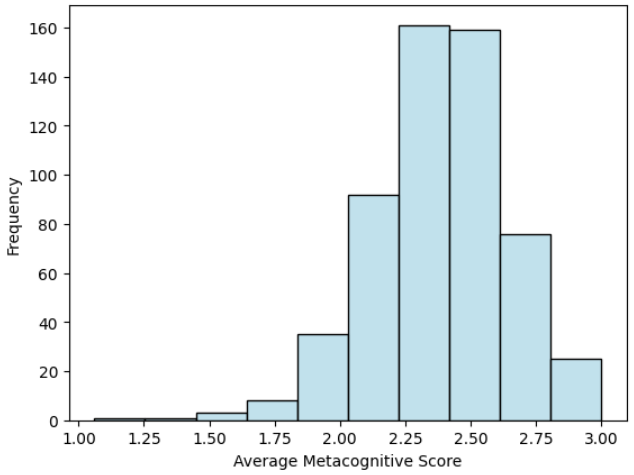


Figure 7: Distribution of average metacognition score - full dataset

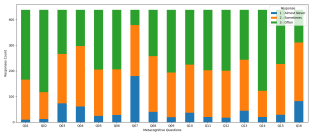


Figure 8: Distribution of students responses for metacognition measuring questionnaire - train split

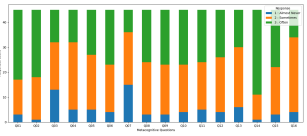


Figure 9: Distribution of students responses for metacognition measuring questionnaire - validation split

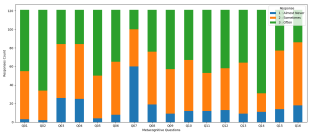


Figure 10: Distribution of students responses for metacognition measuring questionnaire - test split

The distribution of the responses for metacognition measuring questionnaire is shown in figures 8, 9, and 10. And the *Average Metacognition Score* distribution for each split is illustrated in figures 11, 12, and 13 respectively.

3.5.2 Equal distribution of metacognitive profiles across train and test splits. The dataset is split into training, validation, and test splits such that approximately 70%, 10%, and 20% of the instances submitted by each student are allocated to the respective splits. Therefore each split has a similar distribution of metacognition skills which is also similar to the full dataset. The *train, validation, test* splits contains *11,524 , 1460, 3819* instances respectively.

4 Competitive-Level Dataset

We organized a hackathon for final-year and second-year undergraduates to curate a competitive-level dataset. During the hackathon,

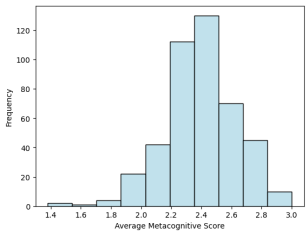


Figure 11: Distribution of average metacognition score - train split

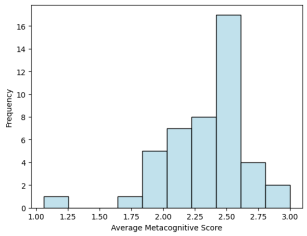


Figure 12: Distribution of average metacognition score - validation split

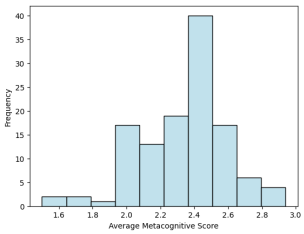


Figure 13: Distribution of average metacognition score - test split

participants solved algorithmic problems, including four easy questions and two medium questions focused on *searching* and *sorting* algorithms. Additionally, we collected data on their metacognitive strategies using the metacognition measuring questionnaire. In total, 45 undergraduates participated, contributing 372 data records. The dataset includes the following categories of questions:

- 2 Introductory Level questions
- 2 Easy questions focused on searching and sorting algorithms
- 2 Medium questions focused on searching and sorting algorithms

The data fields included in the final dataset are as follows:

- **Challenge Name:** The title of the hackerrank challenge
- **Question:** Programming question.
- **Submitted Code:** A code response from a student. The response can be in any of the start/ intermediate/ finish states. The response is in Python programming language.
- **Expected Answer:** The expected correct answer for the programming problem given in Python programming language.
- **Metacognitive vector:** The vectorized representation of the answers to the above 16 questions from the student. This is the student's metacognition profile. The final dataset contains 561 unique metacognitive vectors.
- **Metacognitive feedback:** Annotated metacognitive feedback that aligns with students' solution state and metacognition level.
- **Current Academic Year of Study:** Academic year of studying of the student. First/Second/Third/Final year are the options.

- **Difficulty Level:** Difficulty level of the problem. Easy, medium or hard.
- **Category:** Introductory or sorting or searching.
- **Programming Level:** Self rated programming level of the student. Beginner, intermediate or advance.
- **Accuracy:** The hackerrank metric of accuracy of the submissions to the challenge.
- **Age:** Age of the participant as per year 2025.

The purpose of this dataset is to assess the participants' problem-solving abilities across various levels of complexity, with an emphasis on algorithm design and implementation. This dataset is part of an ongoing effort to gather performance data for competitive programming tasks. For each instance in the dataset, we generated personalized feedback using GPT-4.o mini Utilizing the chain-of-thought prompt in Appendix A. This dataset is yet to undergo expert validation, as outlined in section 3.4.

5 Discussions and Future Work

In future work, we plan to conduct expert validation of the introductory-level dataset with greater statistical significance by involving more experts in the review process. This will help enhance the overall quality of the dataset. We also aim to expand the competitive-level dataset by adding more instances, allowing for a larger and more diverse dataset for analysis. Similarly, we intend to broaden the introductory-level dataset to increase its diversity, improving its applicability across a wider range of problem-solving scenarios and better representing varying levels of metacognitive skills. Another important direction is the validation of the competitive-level dataset. If sufficient human resources are available, we will conduct human evaluations of the annotated metacognitive feedback to ensure its quality and relevance. Alternatively, expert annotation could be used to label additional instances, further improving the dataset's accuracy and value.

6 Conclusion

In this paper, we introduced a novel dataset designed to advance AI-driven educational technologies. This dataset offers a comprehensive and structured collection of student responses, metacognitive skills, and corresponding feedback, enabling researchers to develop and evaluate intelligent tutoring systems with greater adaptability. Through meticulous data collection and preprocessing, we ensured a high quality and diverse representation across a wide range of learning scenarios, contributing to the enhancement of personalized learning experiences.

Despite its significant potential, the dataset has certain limitations, such as lack of human evaluation and validation of each instance, as well as the challenge of ensuring broader generalization across diverse educational settings. Future work could focus on expanding the dataset with additional instances, incorporating a wider variety of student metacognitive capabilities and contextual variations. By leveraging this dataset, researchers can drive meaningful advancements in AI-powered education, fostering more effective, personalized, and responsive learning solutions.

References

[1] Mohammad Munzir Ahanger, Mohd Arif Wani, and Vasile Palade. 2024. sBERT: Parameter-Efficient Transformer-Based Deep Learning Model for Scientific Literature Classification. *Knowledge* 4, 3 (2024), 397–421.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[3] Dileepa Fernando and Chanakya Wijeratne. [n. d.]. Thinking Aloud Protocol Based Self-Report Questionnaire to Measure Metacognitive Skills in Mathematical Problem Solving. ([n. d.]).

[4] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems*. 1–20.

[5] Oka Kurniawan, Cyrille Jégourel, Norman Tiong Seng Lee, Matthieu De Mari, and Christopher M Poskitt. 2021. Steps before syntax: Helping novice programmers solve problems using the PCDIT framework. *arXiv preprint arXiv:2109.08896* (2021).

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

[7] Dastyni Loksa and Amy J Ko. 2016. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM conference on international computing education research*. 83–91.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[9] Abhipraay Nevatia, Soukarya Saha, Sundar Balarka Bhagavatula, and Nikhil Bugalia. 2023. A pre-trained language model-based framework for deduplication of construction safety newspaper articles. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, Vol. 40. IAARC Publications, 387–394.

[10] Sankalan Pal Chowdhury, Vilém Zouhar, and Mrinmaya Sachan. 2024. Autotutor meets large language models: A language model tutor with rich pedagogy and guardrails. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 5–15.

[11] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[12] Minju Park, Sojung Kim, Seunghyun Lee, Soonwoo Kwon, and Kyuseok Kim. 2024. Empowering personalized learning through a conversation-based tutoring system with student modeling. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–10.

[13] Tung Phung, Victor-Alexandru Pădurean, Anjali Singh, Christopher Brooks, José Cambronero, Sumit Gulwani, Adish Singla, and Gustavo Soares. 2024. Automating human tutor-style programming feedback: Leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation. In *Proceedings of the 14th learning analytics and knowledge conference*. 12–23.

[14] Prajish Prasad and Aamod Sane. 2024. A self-regulated learning framework using generative AI and its application in CS educational intervention design. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 1070–1076.

[15] James Prather, Raymond Pettit, Brett A Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM technical symposium on computer science education*. 531–537.

[16] Lianne Roest, Hieke Keuning, and Johan Jeuring. 2024. Next-step hint generation for introductory programming using large language models. In *Proceedings of the 26th Australasian Computing Education Conference*. 144–153.

[17] Gonny LM Schellings, Bernadette HAM van Hout-Wolters, Marcel VJ Veenman, and Joost Meijer. 2013. Assessing metacognitive activities: the in-depth comparison of a task-specific questionnaire with think-aloud protocols. *European journal of psychology of education* 28 (2013), 963–990.

[18] Hamed Taherdoost. 2016. Validity and reliability of the research instrument; how to test the validation of a questionnaire/survey in a research. *International Journal of Academic Research in Management (IJARM)* 5 (2016).

[19] Marcel VJ Veenman and Dorit van Cleef. 2019. Measuring metacognitive skills for mathematics: students' self-reports versus on-line assessment methods. *ZDM* 51, 4 (2019), 691–701.

[20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[21] Doohee You and Samuel Fraiberger. 2024. Evaluating deduplication techniques for economic research paper titles with a focus on semantic similarity using nlp

and llms. *arXiv preprint arXiv:2410.01141* (2024).

A Prompt to Generate Metacognitive Feedback

```
template = """

You are a tutor providing metacognitive feedback
    to students working on programming problems.
    You need to give feedback for their algorithm
    design question answers. You should
    personalize this feedback to their
    metacognition skill level. But you need to
    focus more on identifying the errors and
    optimization in the students code and guide
    them to problem solving without revealing the
    direct answer. Do not provide direct code
    answer for the problem. You should provide
    metacognitive feedback to students working on
    programming problems. The student has answered
     a programming question, and they have a
    metacognitive profile.

*Chain of Thoughts Steps**:
1. **Analyze the Students Answer**: Evaluate if
     the students answer correctly addresses
    the problem requirements. Identify any errors
    or key areas needing improvement.
Problem: {problem}
Student's Answer: {answer}
Expected Correct Solution: {solution}

2. **Compare to Metacognitive Profile**:

**Metacognitive Questions**:
1. I read the question entirely, before I start
    the solving process.
2. I identify and highlight the key requirements,
    inputs, outputs, and constraints of the
    programming task.
3. I rephrase/summarize the question in my own
    words and identify the main points.
4. I create specific input examples and manually
    work through them to reach the outputs to
    understand the problem better before thinking
    about the algorithm.
5. I break down the problem statement into smaller
    , achievable sub-goals before beginning the
    implementation.
6. Before I solve the problem, I estimate/think
    about the nature of the possible algorithm by
    recognizing patterns such as repetition and
    conditional that I would get.
7. I sketch out the algorithm or plan the solution
     before start coding.
8. Every time I revise and execute the designed
    algorithm systematically to reach the answer.
9. I am always vigilant on the implementation
    process to verify that I am on the correct way
     to the solution.
10. I pay attention to avoid negligent mistakes
    during the implementation process.
11. I keep an eye on the problem-solving steps,
    which helps me to verify intermediate results.
12. I always monitor the ongoing program
    implementation process.
13. I check if the algorithm is acceptable and
    compatible with given data constraints.
14. I confirm that the final implementation is
    correct.
15. I refer again to the problem statement and
    check if the implemented solution is
    acceptable for all the given problem
    requirements.
16. I refer to similar problems solved earlier and
     reflect on the accuracy and efficiency of my
    code solution.

The student's responses to these questions are
    recorded as a 16-dimensional vector with
    values of {{1, 2, 3}} corresponding to {{
    almost never, sometimes, often}}. Use this
    vector to understand the student's self-
    reported strengths and areas where they may
    need guidance.
Metacognitive Profile (16-Dimension Vector): {
    metacognitive_vector}
Use the metacognitive profile to understand the
    student's correct metacognitive skill level.
    Identify the skills the student need to
    improve.

3. **Generate Metacognitive Feedback**: Generate
    feedback that addresses errors and
    improvements in the student's code (step 1). \
Focus primarily on the code, but if issues stem
    from weak metacognitive skills, guide the
    student to resolve the problem and enhance
    those skills simultaneously.\
Do not repeat unrelated meatacognition skills. If
    the identified issue/improvement is not
    aligned with the skill, then do not mention it
     in the feedback. \
Only focus on the skills that requires to solve
    the issues in the student's code.

Final output should address the following:
   -DO NOT include code solutions
   -Only output the final metacognitive feedback
       generated under step 3.
   -Should be less than 500 words
   -Do not mention about the questionnnaire.
   -NO NOT REPEAT THE QUESTIONS IN THE STEP 2
       ABOVE.
```

```
   -Focus more on giving specific hints to solve
       the problem, while improving metacognitve
       skills.


### Metacognitive Feedback:
guide how to solve the problem and correct
    mistakes.Output should be a single paragraph
    less than 400 words. Output only the
    metacognitive feedback. Feedback should not
    repeat the metacogntion questions.
"""
prompt_template = PromptTemplate(
    input_variables=["problem", "answer", "
        metacognitive_vector", "solution"],
    template=template
)
```