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 includes students' metacognitive profiles, student submissions for algorithm design problems, and expert-validated metacognitive feedback. MetaFeedback-AD comprises two subsets: MetaFeedback-AD-Introductory, collected from first-year students and containing 16,803 instances across 71 problems and 561 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. 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, 5 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

Personalized learning assistant systems are increasingly important in programming education, where students benefit from tailored guidance to build effective problem-solving skills. Automated feedback systems address this need by providing scalable, real-time support and reducing educator workload. While progress has been made across several domains [12, 14, 20], most programming feedback tools focus solely on code correctness, offering limited personalization beyond the submitted solution [5, 16].

Educational research emphasizes the importance of metacognitive skills—students' ability to plan, monitor, and regulate their learning—for fostering self-regulated learning and long-term success [9, 17, 18]. However, existing programming education datasets largely focus on code correctness and bug detection, offering limited insight into students' learning strategies. For instance, Sahai et al. [21] introduced a dataset with 366 buggy student submissions and GPT-generated feedback, but without metacognitive data. Other datasets, such as CodeContests [7] and APPS [4], support generative models with a wide range of problems and test cases. CodexGLUE [10] and FixEval [3] provide benchmarks for tasks like code completion and program repair. AD 2022 [15] includes over 1,500 solutions to 21 algorithmic tasks but lacks a metacognitive layer. Similarly, CodeNet [19] offers 14 million code samples for 4,000 problems, focusing solely on code output. These resources, while valuable, do not address the metacognitive dimensions critical for personalized, reflective learning.

To address this gap, we introduce MetaFeedback-AD, the first dataset that links algorithm design problems with students' metacognitive self-assessments and expert-validated feedback aligned to those assessments. MetaFeedback-AD includes two subsets: MetaFeedback-AD-Introductory, which is a collection of over 16,000 instances from first-year programming students, and MetaFeedback-AD-Competitive, which is a set featuring more complex problems from upper-level undergraduates. Each instance includes a problem statement, student solution, correct reference solution, 16-item metacognitive self-assessment, a vectorized representation of the responses, and aligned feedback. The dataset consists of disjoint and shared metacognitive profile splits to support varied evaluation settings, enabling researchers to test generalization across unseen learners or adapt to known student types.

By providing this dataset, we aim to facilitate the development of personalized feedback systems that consider both code quality and individual learning strategies, ultimately supporting more effective and reflective programming education.

2 Metacognition Measuring Questionnaire

Metacognition questionnaires assess awareness and regulation of cognitive processes. While Think Aloud Protocols provide rich insights, they are not scalable for large Digital Learning Environments (DLEs) [24]. Self-report questionnaires are easier to use but show only moderate correlation with Think Aloud results, with

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 subgoals 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.

task-specific ones performing better [22]. To address the lack of a programming-specific tool, we adapted and expert-validated a new questionnaire from existing instruments.

2.1 Developing the Questionnaire

We adapted the 16-question metacognition questionnaire from Fernando and Wijeratne [2], originally designed for math problem-solving under Planning, Monitoring, and Evaluation. Questions were revised for clarity and relevance to programming (Table 1). Students rate how often they engage in specific behaviors on a scale of 1 (Almost Never) to 3 (Often), emphasizing observable actions over self-assessment to reduce bias. To ensure domain relevance, the questionnaire was cross-validated with the PCDIT framework by Kurniawan et al. [6], which outlines five non-linear programming phases: Problem Definition, Cases, Algorithm Design, Implementation, and Testing.

2.2 Evaluating the Questionnaire

The modified questionnaire was validated using the method by Taherdoost [23], where four experts rated each item's relevance to the programming domain on a 4-point Likert scale (1 = not relevant to 4 = highly relevant). This process ensured alignment with key aspects of metacognition in programming. Validation results, summarized in Figure 1, were based on the assumption that students' responses honestly reflect their behavior, as supported by the

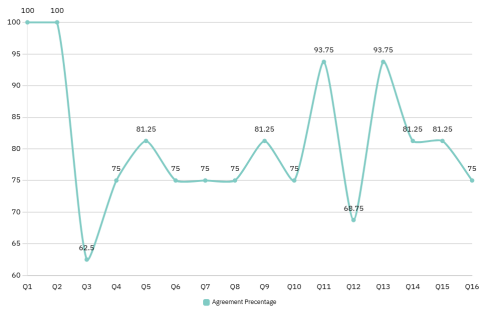

Figure 1: Questions vs Average agreement parentage

Think Aloud Protocol. A weighted agreement percentage was calculated using: Agreement Percentage $= \frac{\sum_{i=1}^{4} \text{ExpertRating}}{\sum_{i=1}^{4} \text{Max Rating (=4)}} \times 100\%$. As shown in Figure 1, expert consensus supported the questionnaire's validity, and it was used in dataset creation.

3 MetaFeedback-AD Dataset

The MetaFeedback-AD Dataset consists of two main components: *MetaFeedback-AD-Introductory*, which includes programming submissions from first-semester students in the Programming Fundamentals module, and *MetaFeedback-AD-Competitive*, which features code submissions from second- and final-year undergraduates solving competitive-level programming tasks. All submissions are written in Python. A detailed summary of the dataset is presented in Table 2.

3.1 MetaFeedback-AD-Introductory

The key data fields in the MetaFeedback-AD-Introductory dataset include; **Question**: An introductory-level programming problem. **Response**: A Python code submission from a student, which may be in a start, intermediate, or finished state. **Right answer**: The expected correct Python solution for the given programming problem. **Q01 to Q16**: Sixteen fields capturing student responses to each question in the metacognition questionnaire. **Metacognitive vector**: A 16-dimensional vector derived from the student's questionnaire responses, representing their metacognitive profile. **Metacognitive feedback**: Annotated feedback that aligns with both the student's code state and their metacognitive level.

3.1.1 Data Collection. The dataset is based on data collected from first-year undergraduates during their first semester in the Programming Fundamentals module. It includes programming problems and student submissions from two assessments (PP1 and PP2) conducted under exam conditions in the Level 3 Laboratory , with access to Moodle content and personal notes allowed, but internet and communication strictly prohibited. To measure metacognitive skills, the questionnaire was administered separately, with 581 out of 1,159 students responding. After data cleaning, the final dataset contains 16,803 instances.

Table 2: Overview of the MetaFeedback-AD dataset

Subset	Number of Instances	Distinct Metacognitive Profiles	Number of Problems
MetaFeedback-AD-Introductory (Total)	16,803	561	71
quad DMP Split (Train / Val / Test)	12,371 / 889 / 3543	438 / 45 / 121	53 / 18 / 18
SMP Split (Train / Val / Test)	11,524 / 1,460 / 3819	560 / 561 / 561	71 / 71 / 71
MetaFeedback-AD-Competitive (Total)	372	43	6

3.1.2 Data Cleaning and Metacognitive Feedback Generation. Initially, two separate datasets were collected: one with student responses to the metacognition questionnaire and another with programming task submissions. These were merged using student IDs, resulting in 21,890 instances. After cleaning—removing nulls, duplicates, and erroneous values—the final dataset contained 16,803 instances. The overview of the cleaning process is illustrated in Figure2.

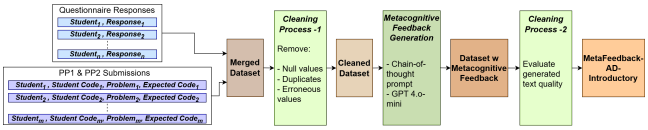

Figure 2: Data Cleaning Process Overview

Each student's responses to the 16 metacognition questions were vectorized into a 16-dimensional metacognitive profile. Metacognitive feedback was generated for each code submission using GPT-4.o mini, informed by both the code and the student's metacognitive profile. A chain-of-thoughts prompting strategy [25] guided the model to: (1) identify code errors and optimizations, (2) interpret the student's metacognitive level, (3) detect metacognitive weaknesses, and (4) provide feedback to improve both code and metacognitive skills. The full prompt is provided in Figure 3.

To ensure dataset quality, a post-processing step, as illustrated in Figure 4 was conducted to remove anomalies and outliers introduced during the generation of the AI-based "Metacognitive Feedback" field. Feedback was generated using both GPT-4.o mini and Mistral-large-latest (v24.11) with the same chain-of-thought prompt (Appendix A). Similarity scores between outputs were computed, and their distribution analyzed. A 95% confidence interval was used to retain only instances within a 5% error margin, improving consistency. Initially, the similarity scores ranged from 0.0130 to 0.9764 (mean = 0.8587, SD = 0.0005), with an IQR-based lower bound of 0.7050. This identified 563 low-similarity instances, which were regenerated using both models. After regeneration, the new lower bound rose to 0.7136, and only 75 instances remained below the initial threshold. Further analysis revealed that 39 of these 75 instances had scores above 0.65, where GPT feedback was coherent and actionable, while Mistral responses were often generic. These 39 instances were retained, and the remaining 36 were manually reviewed and regenerated. To compute similarity, we used SBERT and BERTScore, as these outperform traditional metrics like BLEU [13] and ROUGE [8] in capturing semantic similarity [1, 11, 26].

3.1.3 Expert Validation. We adapted the validation criteria from Roest et al. [20], excluding task-irrelevant or overlapping items to better suit our context. Since that study did not assess 'Metacognitive Alignment', we introduced a new criterion labeled *Metacognitive*. The final set of five criteria used in expert validation is detailed in Table 3. Each of the 30 selected instances is reviewed by an expert using these criteria. Expert response is summarized in Table 4, with votes aggregated per criterion. Preliminary results indicate that the annotated feedback is of high quality. Most instances include all

```
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 Student's Answer**: Evaluate if the student's 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 break down the problem statement into smaller, achievable sub-goals before beginning the implementation.
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.
```

Figure 3: Chain-of-thoughts Prompt

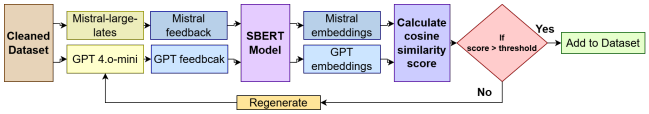

Figure 4: Post-precessing overview

four types of feedback, especially *Tips* and *Explanations*, which are present in every instance. Only six lack a *Compliment*, suggesting the feedback is generally encouraging. All instances are *Personalized* to the student's solution. The *Metacognitive* alignment ranges from moderate to high, with only six flagged for potentially *Misleading Information*. While results are promising, future refinements can further enhance feedback accuracy and effectiveness.

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.

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

3.1.4 MetaFeedback-AD Subsets. We introduce two types of train-validation-test splits for the validated dataset; **DMP**: 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, and **SMP**: Similar 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.

4 Competitive-Level Dataset

We organized a hackathon for second- and final-year undergraduates to curate the competitive-level dataset. Participants solved six algorithmic problems: two introductory, two easy, and two medium questions focused on searching and sorting. Their metacognitive strategies were also captured using the questionnaire. A total of 45 students contributed 372 data records. MetaFeedback-AD-Competitive dataset includes the following additional fields; **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 the year 2025.

The dataset aims to evaluate participants' problem-solving skills across different complexity levels, focusing on algorithm design and implementation. It supports ongoing efforts to collect performance data for competitive programming. Personalized feedback for each instance was generated with the same process under section 3.1.2. Expert validation is pending, as described in Section 3.1.3.

5 Conclusion

We introduced MetaFeedback-AD, the first dataset to combine structured metacognitive self-assessments with student code submissions for algorithm design tasks. It enables personalized feedback generation based on both solution correctness and metacognitive profiles, addressing a key need in intelligent tutoring systems. MetaFeedback-AD comprises two subsets—Introductory and Competitive—capturing varying student skill levels and problem complexity. The dataset includes 17,175 code–feedback pairs across 604 profiles and 78 questions. While current validation covers only a subset, future work will expand expert review across all feedback. Broader data collection from diverse learners and algorithm problems will improve representativeness. By releasing MetaFeedback-AD publicly, we aim to support research in adaptive, learner-aware educational technologies for computing.

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

[3] Md Mahim Anjum Haque, Wasi Uddin Ahmad, Ismini Lourentzou, and Chris Brown. 2022. FixEval: Execution-based Evaluation of Program Fixes for Competitive Programming Problems. *arXiv preprint arXiv:2206.07796* (2022).

[4] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938* (2021).

[5] 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.

[6] 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).

[7] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

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

[9] 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.

[10] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664* (2021).

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Fynn Petersen-Frey, Marcus Soll, Louis Kobras, Melf Johannsen, Peter Kling, and Chris Biemann. 2022. Dataset of student solutions to algorithm and data structure programming assignments. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. 956–962.

[16] 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.

[17] 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.

[18] 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.

[19] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. 2021. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655* (2021).

[20] 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.

[21] Shubham Sahai, Umair Z Ahmed, and Ben Leong. 2023. Improving the coverage of gpt for automated feedback on high school programming assignments. In *NeurIPS'23 Workshop Generative AI for Education (GAIED). MIT Press, New Orleans, Louisiana, USA*, Vol. 46.

[22] 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.

[23] 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).

[24] 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.

[25] 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.

[26] 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).