

A Comprehensive Survey on Misconception Diagnosis in AI-Assisted Programming Education: Rule-Based and LLM-Based Techniques, Accuracy-Cost Trade-offs, and Open Challenges

Amarsinh Varpe¹, Chaitanya Sonawane², Tanmay Dhole³, Vrushali Jagtap⁴, and Sai Kannadkar⁵
^{1,2,3,4,5} Department of Computer Science and Engineering, Sandip University, Nashik, Maharashtra, India

Abstract - A longstanding goal of programming-education tooling is to do more than mark an answer wrong and identify the specific misconception the novice programmer might have in their program. Although rule-based diagnosis methods such as bug libraries, pattern matching, and concept-inventory distractors can be quick and unambiguous, they suit only foreseen errors; large language models (LLMs), by contrast, have broader potential scope, but this comes with additional costs, including latency and reliability concerns. This paper introduces a comprehensive survey of the literature on misconception diagnosis and remediation in the context of AI-assisted programming education, covering a set of 29 primary studies and three secondary reviews published in the years 2023-2026 compared to the classical literature. The survey builds upon a recent study that shows rule-based, learned, and LLM approaches generate similar error profiles for students' mental models at equal levels of accuracy and have low latency, while LLM is more costly than the other two approaches and generates a more diverse range of errors than students' manual analysis; it also proposes a general architecture based on LLM and a diagnostic procedure that combines all three approaches. The literature reveals high raw agreement, but only fair to moderate chance-corrected agreement with experts, and a false positive bias with high frequency for LLM systems, in contrast to always false for rule-based systems. Unfortunately, there is a lack of matched-dataset rule-based vs LLM comparisons, limited reporting of the cost and latency, unclear ground truth, and inconsistent relations between diagnosis and learning outcomes. Directions for further research are given. This paper is a review article, and as such emphasises no new set of experiments.

Index Terms - programming education, misconception diagnosis, mental models, large language models, rule-based diagnosis, intelligent tutoring systems, automated feedback, survey.

I. INTRODUCTION

Even for novices, learning how to program can be extremely challenging, and a significant part of this challenge is not in learning a new programming syntax, but because of long-lasting incorrect models (also known as misconceptions) of what code really is and does, that are difficult to unlearn [1, 2]. Any student who thinks "equals" sets an algebraic equation, that a loop resets all its variables with each iteration, or that a pointer is the object to which it points will make the same kind of mistake over a variety of different problems. Since the first intelligent tutoring systems [3, 4], it has been a goal of programming-education tooling to detect the root cause of the

error (the misconception) rather than just the compiler error and then to remediate that specific misconception.

The diagnostic part has been mostly covered by rule-based systems: bug libraries, abstract-syntax-tree (AST) pattern matcher, and hand-curated catalogues of code and/or answer patterns used for specific misunderstandings [3] [5]. Validated concept inventories further involved creating choices that would reflect each misconception uniquely [6, 7, 8]. These are speedy, predictable, and explainable machines... and only as good as whatever patterns their creators foretold. Large language models (LLMs) have added a second paradigm in which reasoning over code, free-text explanations and context become the primary way to address a task rather than matching fixed patterns, since late 2022. In a relatively short time, the computing-education venues have released numerous research papers related to LLM-based tutors, feedback generators, and misconception analysers [9, 10, 11] and studies that reported some of the limitations of these tools, such as providing false explanations and detecting errors in correct code [12, 13, 14].

Previous work consists of two types of reviews: systematic reviews, which document the use of LLMs across computing education broadly [14, 15] and systematic reviews of automated grading and feedback tools [16, 17]. None of them has been structured around the most important question to any designer of a misconception-diagnosis system: For a given misconception, what is the accuracy, latency, and financial cost of a diagnosis achieved with a rule-based system vs. one powered by an LLM; how can they be combined? This survey is to answer that question. As background for this report, we rely on the most thorough recent study of how LLM tools can be used as misconception analysers, which reports side-by-side reliability and cost: the one by Gallego-Durán et al. [13]. We adopt the design and its limitations of this effectively as a baseline for organising the broader literature.

There are three research questions for this survey. RQ1: What are the diagnostic techniques (rule-based, non-LLM learned, and LLM-based) that have been used to understand programming misconceptions and how are they organised? RQ2: What are the reported accuracy, error profile, latency and cost of these techniques? RQ3: What other areas are missing, and what would be preferable avenues to go into to rectify these?

The contributions of this survey are: (i) a structured synthesis of 29 primary studies and 3 programming-specific secondary reviews published between 2023 and 2026 against the classical

misconception literature; (ii) a detailed analysis of a base paper [13] for use as a reference design for LLM-based misconception diagnosis; (iii) a per-study comparison that explicitly records whether a study reports cost or latency, and a three-way comparison of rule-based, learned, and LLM-based methods; (iv) a synthesized general architecture and misconception diagnostic workflow from the reviewed systems; and (v) a prioritized set of research gaps and directions. This is a survey paper, with the characteristic of this type of paper being no new system introduced and no new data generated.

The rest of the paper is organised as follows: Section II presents the methodology of the survey and the literature review, which starts with the base paper. In Section III, the reviewed studies are compared, analysed and discussed. The research gaps and challenges are identified in Section IV, followed by the future research directions in Section V, and finally, Section VI concludes.

II. BACKGROUND AND LITERATURE REVIEW

A. Survey Methodology

In contrast, an extensive enumeration would have been more appropriate for a slower subfield, and a structured narrative review protocol was used with the explicit inclusion criteria that were best suited. The keywords "programming education", "misconception", "mental model", "intelligent tutoring", "large language model", "automated feedback", "automated grading", and "knowledge tracing" were used to retrieve candidate studies from the ACM Digital Library, IEEE Xplore, SpringerLink, and arXiv during 2026. Backwards and forward snowballing of existing reviews [14] – [15] and [17] were used to supplement keyword search.

The study was required to (i) have been published, in some peer-reviewed conference or journal, between Jan 1, 2023 and mid-2026; (ii) report an empirical evaluation, or an evaluated system, of real practitioner student code, answers, or classroom deployments; (iii) have dealt with programming or computer-science education; and (iv) have been published in a peer-reviewed conference or journal. Opinion pieces without an evaluated system and/or dataset were not included. The titles & abstracts were then scored for consistency with these criteria, and then all the other titles were evaluated on a full-text basis. The final corpus consists of 29 main studies (individualised in Table I) and 3 secondary reviews of the programming-specific ones [16], [17], [14]. Classical works before 2023, preprints (e.g., [18], [15] and [19]), as well as general-education surveys, are cited simply as context and do not appear within the corpus.

B. Historical Foundations and Misconception Catalogues

The brand new misconception diagnosis in programming can have a history of about four decades. In PROUST, the programmer's intention is diagnosed by checking the match between a novice program to be diagnosed and a plan library and known bugs [3], and LISP Tutor used model tracing to find the match between each student step and correct/ "buggy" production rules [4]. These systems outline the architecture that has recurred in our times: an expert/domains model, the student model, the pedagogical model, and the interface.

A parallel team set about documenting misconceptions. Kaczmarczyk et al. isolated certain common misunderstandings regarding memory, variables, and loops [8], and Qian and

Lehman examined the broader evidence of novice errors [1], while Sorva suggested that many of these misunderstandings can be understood as problematic notional machines [2]. The purpose of the inclusion of "wrong" answers, or "lures", in validated questionnaires as in FCS1 and its replication SCS1 is deliberately to create options that will appear to be the correct answer, but actually target specific misconceptions [6, 7], and the Progmiscon inventory has already collated language-specific misconceptions in a reusable form [5]. Distractors/patterns in these catalogues naturally correspond to one misconception label: one and only one distractor-pattern to one misconception label.

C. Base Paper: Evaluating GPT as an Automated Mental-Model Analyser

Weizmann and Gilbert [6] characterise coding as a cognitive task that is approached by students' mental models. Coding is a cognitive task that can be approached by students' mental models, as emphasised by Weizmann and Gilbert [6] and evaluated by Gallego-Durán et al. [13] in the context of automated analysers, namely GPT-4 and GPT-5. The method is iterative, following six steps: Make an assumption about incorrect mental models, design an open-ended discriminating question, have students answer the questions, code and tally each answer for incorrectly assumed models, determine the ranking of the models and questions, and repeat. The study builds on this by taking this cycle into a hybrid human-AI cycle, where the first pass through coding is performed by an LLM, which also generates new hypotheses that are reviewed and curated by instructors.

This dataset consists of 1500 responses to 24 open-ended questions concerning variables, types, char type, the use of pointers, parameters and memory, provided by 50 fourth-year students responding to 27 mental models hypothesised in coding their answers, which yields a total of 5300 binary judgments. One question/one student answer for each response, these optimised prompts resulted in an 82.7% raw agreement with faculty (with Cohen's kappa of 0.38, far below the 0.60–0.80 that the authors suggest is what the answerer and subject expect between themselves). (GPT-4: 66.9% raw; 0.18 Cohen's kappa). Overall, there was great agreement between the two raters for "no evidence" (74.7%) detection, and few joint positive detections (8.1%). A mistake by faculty was indicated by both models, with a false-positive bias of 28.1% for GPT-4 and 11.6% for GPT-5, resulting in precision of 0.24 and recall of 0.65 and 0.59, respectively. Importantly for this survey, it also states cost and time: For this survey, the study shows that the analysis of all responses took roughly 36 h and US\$19.06 using GPT-4, versus 4.7 h and US\$12.51 using GPT-5, which is around 14.2 s per query.

The authors suggest that, having a fair-to-moderate kappa, LLMs could be used as initial analysers in a blended system (along with systematic faculty review) but that it should be considered as a "production-ready" framework rather than used without it. They also recognise several limitations, which are directly relevant to this survey: single-institution, single-language (C++), single-cohort; no inter-human reliability could be computed; only proprietary GPT models; but most importantly for us, no comparison to a rule-based or non-LLM version of the same baseline on the same data.

D. Rule-Based and Learned (Non-LLM) Diagnosis

The best standards against which to compare LLM approaches should always be the deterministic approach. Large amounts of student code can be analysed for semantic flow by the CFlow model, thus informing the teacher on the deviations of some student's control or data flow from the expected solutions by using an LLM [20]. We can look up an observed response of the student on a misconception label from a Concept-inventory Distractor [6] or Curated Inventory like Progmiscon [5]. These works are inexpensive and relatively quick to implement and have the advantage of being monitored, but are limited in scope by their designers' expectations of use.

A middle level is based on learned (but non-generative) models. Hoq et al. implemented an interpretable AST-based attention neural network to detect "logical errors" in student code and to trace recurring patterns of errors over problems as a signature of what may be a misconception [21], and they later built a scalable model that was capable of detecting logical errors based on information about the correctness of the student's code [22]. These are less expensive and more legible than LLMs, but require training data and are more general than hand-written rules, but don't come with a natural language misconception description.

E. LLM-Based Misconception and Error Diagnosis

There's an emerging body of research assessing the effectiveness of LLMs in the diagnostic task. While aspects of their answers for real novice help requests were often plausible, they were not always pedagogically appropriate, as suggested by Hellas et al. [23]. MacNeil et al. [11] compared the ability of GPT-3, GPT-4 and CS1 students at identifying bugs by logic errors, and surprisingly found that the models had significantly better accuracy on incorrect code, but did not perform as well on correct code. It was common for ChatGPT to reliably find common types of errors made by beginners, but inconsistently flagging the idiosyncratic type of code [24]. Kiesler et. al. demonstrated the usefulness and sometimes incorrectness of the LLM's errors [12]. In addition to coding, Oli et al. found that fine-tuned models were superior to zero- and few-shot prompting in identifying students' gaps and misconceptions related to their natural-language code explanations [19]. The former property—that it forms part of this cluster and has chance agreement corrected—sets it apart from a number of different papers, while the latter property—that is, in the cost accounting it is explicit—is what distinguishes it from a majority of papers.

Combined, these studies reveal a uniform set of errors: LLMs can identify misconceptions not explicitly specified in any particular set of rules, or they can suggest misconceptions that the instructors did not envision [13]; however, they seem to over-number, "calling to attention" misconceptions or errors in the correct work or ambiguous samples [11], [13].

F. LLM Tutors and Misconception-Targeted Remediation

Pedagogical diagnosis is significant only when it leads to targeted remediation. CodeHelp is an LLM wrapped into "guardrails" that explain problems without producing the (full) solution [9], and a study of its logs identified the time and manner of student help requests [25]. Alternatively, dcc-help was deployed in a twelve-week C course, where students found the assistance from most of the responses to the compiler-

diagnosed errors to be very informative and helpful—and gathered a mixture of deterministic (compiler) output, combined with generative (LLM) explanation. Specifically, Niousha et al. based an LLM tutor on an explicit list of common misconceptions, and observed that even basic prompting techniques resulted in tutor responses becoming more pedagogically appropriate and focused on misconceptions [27]. Kerslake et al. added prompting practice to introductory courses upstream of the tutor [28], and Yang et al. considered novice users of an AI debugging tutor and the relationship between how novice users seek help and their perceived learning levels [29].

Delving into what's going on with code elaboration and synthesis highlights the importance of focusing on understanding, not on tasks. Although novices often found the LLM explanations easier to understand [37], [38], the use of generated code also introduces variability in how well students can debug on their own [39], [40], [41], [42], leading Prather et al. to say that there is "widening variability in how well students can debug on their own" [43].

G. Adjacent Areas: Knowledge Tracing and Automated Grading

There are two adjacent strands providing components that a misconception-diagnosis system requires. Knowledge-component discovery (KCD) is now automated or hand-labelled [37] (or with LLMs [38]), and many problems that are small and separately traced lead to better predictions of struggle [39], so the much broader literature around knowledge-tracing is reviewed in [18]. With automated grading, there is almost universal consensus among teachers on the feedback and grading that the LLM provides [40]–[43], when compared to what teachers do in baseline before the LLM [16]–[17]. Per-submission latency and API cost are not generally reported or quantified in this strand, other than by Lagakis et al., who report reduced grading time by instructors when using their system [43].

H. Generalised Architecture and Diagnostic Workflow

On the one hand, the systems examined have a common structure in their illustrated design—represented in Fig. 1—and on the other hand, they appear in the same order in the text. It is not claimed as "original" but is developed from what has been written. The input for a submission is preprocessed to produce tokens or an AST (which could be a program of code or free text). An LLM-based path that reasons about the input and contextual information [9, 10, 13] and a rule-based path that searches the input, followed by a pattern, distractor, and/or misconception library [3, 6, 5, 20] are both reported to occur after a diagnostic engine. In between are learned (non-LLM) models [21, 22]. As a result, this label is presented to a feedback generator, sometimes with precautions to prevent solutions from leaking off of [9],[27] and updates a student or knowledge-component model [37],[38].

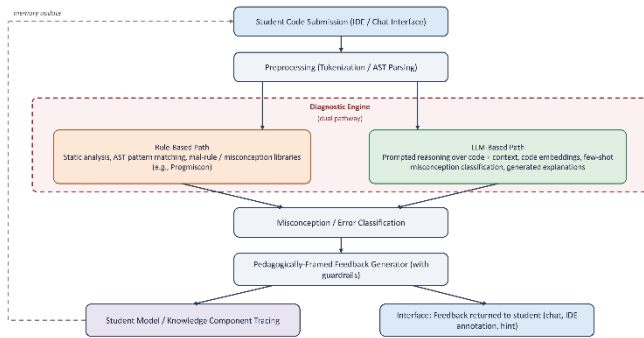


Fig. 1. Generalised architecture synthesised across the reviewed systems, showing the dual rule-based / LLM-based diagnostic pathway.

In the diagnostic step, the concrete becomes real, as seen in Fig. 2. The submission is initially matched against the rule library to get a quickly determined label. If not, the system defaults to LLM reasoning; if the LLM diagnosis is used or escalated to generic or tracked to human diagnosis, a confidence and/or a guardrail check is performed. It is similar to the base paper, which integrated a human-AI handling system where LLM output was monitored by instructors [13] and deployed tutors had guardrails [9], [10].

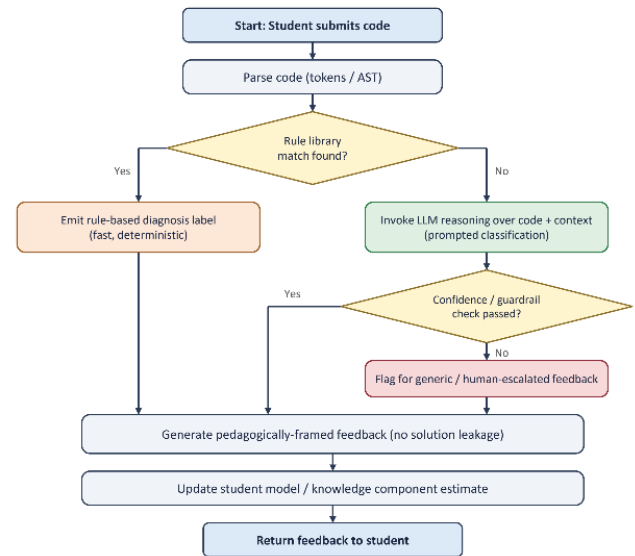


Fig. 2. Generalised diagnostic workflow abstracted from the reviewed systems: rule-first, LLM-fallback, with a guardrail/confidence check before feedback is returned.

III. COMPARISON AND ANALYSIS OF EXISTING STUDIES

The reviewed primary studies are compared to one another by diagnostic paradigm in tabular form (Table I), task, evaluation basis, main reported finding, and the report of cost and/or latency. Table II summarises the three paradigms of diagnostics on those dimensions that impact deployment.

TABLE I. PER-STUDY COMPARISON OF REVIEWED PRIMARY STUDIES

Ref.	Paradigm	Task	Evaluation basis	Main reported finding	Cost/latency as outcome
[13]	LLM (hybrid review)	Mental-model detection, C++ free text	5,300 binary judgments vs. 4 faculty	GPT-5: 82.7% agreement, $\kappa=0.38$, $P=0.41$, $R=0.59$; GPT-4: $\kappa=0.18$	Yes
[21]	Learned (AST attention NN)	Logical-error / misconception localisation	Correct vs. incorrect programs	Highlighted subtrees align with logical errors (preliminary)	No
[22]	Learned (AST attention NN)	Logical-error identification at scale	Student programs, correctness labels	Accurate error localisation without rubrics or tests	No
[20]	Rule/program analysis	Semantic flow analysis	Instructor-facing evaluation at scale	Surfaces divergent flow patterns without LLMs	No
[11]	LLM	Logic-error detection	GPT-3/4 vs. CS1 students	High detection on faulty code; weaker on bug-free code	No
[23]	LLM	Responses to help requests	Expert analysis of responses	Often plausible, not always pedagogically appropriate	No
[24]	LLM	Misconception/error identification	Experience report on novice code	Familiar errors found; less consistent on unusual code	No
[12]	LLM	Formative feedback	Feedback quality analysis	Helpful but sometimes inaccurate feedback	No
[27]	LLM + misconception catalogue	Misconception-aware tutoring	Classroom student-tutor interactions	Catalogue-grounded prompting improves pedagogical fit	No
[9]	LLM + guardrails	Help-seeking support	Multi-course deployment	Positive reception; solution leakage contained	No
[25]	LLM + guardrails	Help-seeking analysis	Interaction logs	Characterises when and how students ask for help	No
[10]	LLM + guardrails	Programming assistant	12-week C course	Most responses judged accurate and helpful	No
[26]	Hybrid (compiler + LLM)	Compiler-error explanation	Classroom deployment	Context-aware explanations of compiler errors	No
[29]	LLM	AI debugging tutor	Observational study	Help-seeking behaviours and perceived learning	No
[28]	LLM (prompting skill)	Prompting in CS1	Course integration	Prompting practice supports LLM tool use	No
[30]	LLM	Code explanation	Student vs. LLM explanations	LLM explanations often rated clearer	No
[31]	LLM	Code explanation (e-book)	Student engagement	Explanations used; reliance risk	No
[32]	LLM code generation	Novice task completion	Controlled study	Faster completion; weaker independent debugging	No
[33]	LLM code generation	Novice usage patterns	Self-paced study	Varied usage; over-reliance risk	No
[36]	LLM (generative AI)	Novice benefits/harms	Qualitative study	"Widening gap" between learners	No
[34]	LLM (Copilot)	Solving CS1 problems	Problem set	Solves most simple problems	No
[35]	LLM (Copilot)	Novice usability	Usability study	Difficult to form a mental model of the tool	No
[37]	Learned	KC discovery	Agreement with KC labels	Reduces manual annotation	No
[38]	LLM	KC extraction, CS1	Agreement with experts	Reduces manual annotation	No
[39]	Statistical (learning curves)	Knowledge tracing granularity	Course data re-analysis	Many small problems improve estimates	No
[40]	LLM	Feedback generation	Feedback quality criteria	Broadly comparable to instructor feedback	No
[41]	LLM	Short-answer grading (SE)	Agreement with graders	Practically useful agreement	No
[42]	LLM	Exam-code grading	Qual. + quant. consistency	Usable with inconsistencies	No

[43]	LLM	Coding-exercise grading	Grading pipeline evaluation	Reduced grading time, quality maintained	Partial (time)
------	-----	-------------------------	-----------------------------	--	----------------

P = precision, R = recall, κ = Cohen's kappa, KC = knowledge component. Cost/latency as outcome: whether per-diagnosis monetary cost or response latency is quantified as a reported outcome of the study.

There are three patterns that stand out in Table I. First of all, only one primary paper (the base paper [13]) quantifies the monetary cost and the latency per diagnosis. Secondly, only one paper (partially [43]) reports quantitatively on the time taken to grade the pictures per diagnosis. Although perks and transfer factors are important aspects to consider when determining if a technique can be realistically applied to a large course, the primary metrics reported are accuracy and helpfulness. Second, there is no evaluated study that compares a rule-based or concept-inventory diagnoser and a diagnoser based on an LLM on the same data set. Comparisons are either LLM vs students [11] or LLM vs faculty [13] or LLM vs LLM [13] [19], and again, any advice on which paradigm is "better" would necessarily have to have been derived from comparing across papers. Third, the evidence is very weak: Out of the diagnostic tools examined, a few that are sometimes reported as rule-based have indeed been learned [21, 22] or LLM-assisted [26].

TABLE II. COMPARISON OF DIAGNOSTIC PARADIGMS

Dimension	Rule-based / inventory	Learned (non-LLM)	LLM-based
Examples	[3], [6], [5], [20]	[21], [22], [37]	[13], [11], [27], [19]
Coverage	Anticipated patterns only	Patterns in training data	Broad; can propose new misconceptions
Typical error	Misses novel errors	Needs labelled data; limited explanation	False positives / over-detection
Explainability	High (rule fires)	Medium (attention)	Fluent text; faithfulness uncertain
Cost/latency	Near-zero; ms	Low after training	Per-call API cost; seconds per query
Determinism	Deterministic	Deterministic at inference	Non-deterministic; prompt-sensitive

A key difference between the paradigms is that the base paper and MacNeil et al. both agree it is behavioural. MacNeil et al. show that humans made mistakes with buggy code substantially more often than their LLMs, but students were much better than these LLMs at identifying code without bugs; and the better of these two LLMs was the most likely to mark one without bugs [11]. Gallego-Durán et al. report the same over-detection in free-text items, and claim that the main reason for the improvement between generations is due to reducing false-positives not to increasing true-positives [13]. By construction, a rule-based or inventory diagnoser will not fire if the input does not match the known patterns, and so it won't be able to detect new misconceptions, but it won't invent them.

The cost dimension is also oriented in this direction. While it is possible to get LLM-based analysis to be cheap individually, each individual call to an LLM comes at a cost and still has a latency of seconds, while a rule lookup is cheap and has an instantaneous latency. While LLM analysis of a full course cohort is now affordable in absolute terms, every analysis performed still costs something and takes seconds. Meanwhile, a rule lookup can be done with very few seconds of latency and for little to no cost. However, its Hallett-Rand agreement is not very high and is only quite reasonable, which means that the more affordable a system is, it is not automatically reliable. The most significant deployments reported are those that couple techniques: an LLM for coverage/explanation via natural-language explanations, followed by a rule-based check,

guardrail and/or human review to mitigate over-detection [9] [27] [13].

IV. RESEARCH GAPS AND CHALLENGES

A. No Matched-Dataset Rule-Based vs. LLM Comparison

There is not a single study that runs both a rule-based and a concept-inventory diagnoser on the same collection of students' answers annotated accordingly. The base paper explicitly benchmarks only in terms of faculty GPT generations [13]. Hence, it is still unclear which of the techniques is better in terms of accuracy, cost-efficiency and speed within the same misconception category.

B. Cost and Latency Rarely Reported

However, apart from the base paper [13] and a partial time paper [43], neither cost nor latency is explicitly reported as a primary metric, even in grading papers, where its use as a comparison with the near-zero-cost test-case autograders is mentioned, though not very explicitly [40, 41, 42].

C. Reliability, Over-Detection, and Faithfulness

Very high agreement when the great majority of cases are negative doesn't reflect the reliability of the LLM. The level of agreement for the base paper result is very low (82.7%), with a Kappa coefficient of 0.38 [13], and this is also observed in both other studies [11] and [19]. The form of quantitative risk is that of "fluent but unfaithful feedback" [12] [24], which is not the same as a blank firing of a rule.

D. Uncertain Ground Truth

Numerous studies take faculty labels as gilded standards. The base paper identified that inter-human kappa was not computed for the base paper and claimed that in previous research, disagreements between the LLM's reasoning and the initial human coding were found in approximately half of the cases, where the LLM's reasoning was considered to be more defensible than the initial human coding [13]. Measured accuracy without validated misconception inventories [6, 7] to anchor is hard to interpret.

E. Diagnosis Disconnected from Remediation and Learning Outcomes

Most diagnostic studies stop at the label. Most of the diagnostic studies are limited to the label. Research into misconception-based remediation [27] or studies on long-term effects of learning [32], [36], and [29] is short-term and is not always associated with validating diagnosis of a misconception, making it difficult to determine if remediation that targets a misconception is superior to a generic lesson.

F. Narrow Coverage

Evidence on a single-institution CS1 cohort, some languages (Python, Java, C, and C++), English or Spanish instruction, and specific proprietary models [13], [11], [10]. Cost and student-data privacy considerations are important for open-weight models, which are not measured in many cases, and there is only approximate cross-study comparison due to the many different metrics employed [17, 14].

V. FUTURE RESEARCH DIRECTIONS

A. Matched-Dataset Benchmarks

A benchmark that includes a misconception catalogue validated using students' responses to identify misconceptions, and that uses rule-based and/or learned and LLM-based diagnosers to arrive at the same catalogue on the same set of student responses and reports the accuracy, kappa, precision, recall, latency, and cost per diagnosis side by side would directly answer the main open question. For replication, such datasets can be published from the public materials of the base paper [13].

B. Principled Hybrid Routing

The gist of the proposed design (Fig. 2) is to use a rule-first, LLM-fallback approach, which is currently done ad hoc. It could be possible to use a combination of model and rules to improve coverage and precision, as well as cost, since a response may only be sent to the LLM if there is no match, or if it only indicates low confidence. It might be feasible to combine the LLM with the rules in a way that the rules pre-qualify when possibly correct, and the LLM vetoes when probably mistaken.

C. Faithfulness and Calibration Benchmarks

However, generated diagnoses should be tested against shared benchmarks to determine if the diagnosis typically identifies the actual error in the actual submission, and shared benchmarks should report chance-corrected agreement (kappa), not just agreement.

D. Diagnosis-to-Remediation Studies

This is through controlled studies that compare the effectiveness of misconception-targeted remediation to generic remediation, and measure the rate of misconception corrections, particularly based on misconception-aware tutoring [27] and the potential for over-reliance [36].

E. Broader and Open Evaluation

Other languages, languages over multiple institutions, in-language instruction, and the open-weight models that can be executed on local machines should also be evaluated as recommended in the base paper [13].

VI. CONCLUSION

This survey examined AI-powered diagnosis of misconceptions in programming education between 2023 and 2026, with the previous existing base paper [13] serving as a framework for evaluation of algorithms, such as GPT-4 and GPT-5, for measuring student mental models as 'mental model analysers'. The study reflected 40 years of research on misconception diagnosis for programming education, using the base paper [13] as a foundation to evaluate the ability of mental model analysers, such as the algorithms GPT-4 and GPT-5. It's not as though there's any evidence or literature that surrounds the notion that LLMs have superseded rule-based diagnosis. While LLM models provide extensive coverage, they can suggest new misconceptions and, although they are now widely accessible, they consistently produce false positive responses and have only fair-to-moderate chance-corrected accuracy over experts. Where they are applicable, the rule-based and inventory approaches are fast, deterministic and costless. The

most significant missing pieces are the lack of matched dataset comparisons between the two paradigms, limited reporting of cost and latency data and the poor connection between diagnoses and measured learning gains. Rather than further development of an autonomous set of tutoring interfaces, closing these gaps in evaluation seems to be the defining next milestone in the field.

REFERENCES

- [1] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, Art. no. 1, 2017.
- [2] J. Sorva, "Notional machines and introductory programming education," *ACM Trans. Comput. Educ.*, vol. 13, no. 2, Art. no. 8, 2013.
- [3] W. L. Johnson and E. Soloway, "PROUST: Knowledge-based program understanding," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 3, pp. 267–275, Mar. 1985.
- [4] J. R. Anderson and E. Skwarecki, "The automated tutoring of introductory computer programming," *Commun. ACM*, vol. 29, no. 9, pp. 842–849, 1986.
- [5] L. Chiodini, I. Moreno Santos, A. Gallidabino, A. Taffliovich, A. L. Santos, and M. Hauswirth, "A curated inventory of programming language misconceptions," in *Proc. 26th ACM Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, 2021, pp. 380–386.
- [6] A. E. Tew and M. Guzdial, "The FCS1: A language-independent assessment of CS1 knowledge," in *Proc. 42nd ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2011, pp. 111–116.
- [7] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, validation, and use of a language-independent CS1 knowledge assessment," in *Proc. ACM Conf. Int. Comput. Educ. Res. (ICER)*, 2016, pp. 93–101.
- [8] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proc. 41st ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2010, pp. 107–111.
- [9] M. Liffiton, B. E. Sheese, J. Savelka, and P. Denny, "CodeHelp: Using large language models with guardrails for scalable support in programming classes," in *Proc. 23rd Koli Calling Int. Conf. Comput. Educ. Res.*, Koli, Finland, 2023.
- [10] M. Kazemitabaar et al., "CodeAid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs," in *Proc. 11th ACM Conf. Learn. @ Scale (L@S)*, Atlanta, GA, USA, 2024.
- [11] S. MacNeil et al., "Decoding logic errors: A comparative study on bug detection by students and large language models," in *Proc. 26th Australas. Comput. Educ. Conf. (ACE)*, Sydney, Australia, 2024.
- [12] N. Kiesler, D. Lohr, and H. Keuning, "Exploring the potential of large language models to generate formative programming feedback," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, College Station, TX, USA, 2023.
- [13] F. J. Gallego-Durán, P. Compañ-Rosique, and C. J. Villagrà-Arnedo, "Evaluating GPT as automated analyzer for detecting students' erroneous mental models in programming education," *Univ. Access Inf. Soc.*, vol. 25, Art. no. 51, 2026, doi: 10.1007/s10209-026-01320-z.
- [14] N. Raihan, M. L. Siddiq, J. C. S. Santos, and M. Zampieri, "Large language models in computer science education: A systematic literature review," in *Proc. 56th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE TS)*, Pittsburgh, PA, USA, 2025.
- [15] M. Zhu, L. Xu, et al., "A systematic review of research on large language models for computer programming education," *arXiv:2506.21818*, 2025.
- [16] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Machine learning-based automated grading and feedback tools for programming: A meta-analysis," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, 2023.
- [17] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Trans. Comput. Educ.*, vol. 24, no. 1, pp. 1–43, 2024, doi: 10.1145/3636515.
- [18] G. Abdelrahman, Q. Wang, and B. Nunes, "Knowledge tracing: A survey," *ACM Comput. Surv.*, vol. 55, no. 11, pp. 1–37, 2023.

- [19] P. Oli, R. Banjade, A. M. Olney, and V. Rus, "Can LLMs identify gaps and misconceptions in students' code explanations?," arXiv:2501.10365, 2025.
- [20] A. G. Zhang, X. Tang, S. Oney, and Y. Chen, "CFlow: Supporting semantic flow analysis of students' code in programming problems at scale," in Proc. 11th ACM Conf. Learn. @ Scale (L@S), Atlanta, GA, USA, 2024.
- [21] M. Hoq, J. Vandenberg, B. Mott, J. Lester, N. Norouzi, and B. Akram, "Towards attention-based automatic misconception identification in introductory programming courses," in Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE) V. 2, Portland, OR, USA, 2024, pp. 1680–1681, doi: 10.1145/3626253.3635575.
- [22] M. Hoq et al., "Automated identification of logical errors in programs: Advancing scalable analysis of student misconceptions," in Proc. 18th Int. Conf. Educ. Data Mining (EDM), 2025.
- [23] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, and J. Sorva, "Exploring the responses of large language models to beginner programmers' help requests," in Proc. ACM Conf. Int. Comput. Educ. Res. (ICER), Chicago, IL, USA, 2023.
- [24] H. L. Fwa, "Experience report: Identifying common misconceptions and errors of novice programmers with ChatGPT," in Proc. 46th Int. Conf. Softw. Eng.: Softw. Eng. Educ. Training (ICSE-SEET), 2024.
- [25] B. Sheese, M. Liffiton, J. Savelka, and P. Denny, "Patterns of student help-seeking when using a large language model-powered programming assistant," in Proc. 26th Australas. Comput. Educ. Conf. (ACE), Sydney, Australia, 2024.
- [26] A. Taylor, A. Vassar, J. Renzella, and H. Pearce, "dcc --help: Transforming the role of the compiler by generating context-aware error explanations with large language models," in Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE), Portland, OR, USA, 2024.
- [27] R. Niousha, S. Boatright Smith, A. O'Neill, J. D. Zamfirescu-Pereira, J. DeNero, and N. Norouzi, "Misconception-aware LLM programming tutor: Lessons learned from student-tutor interactions," in Proc. 57th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE TS), 2026.
- [28] C. Kerslake et al., "Integrating natural language prompting tasks in introductory programming courses," in Proc. ACM Virtual Global Comput. Educ. Conf. (SIGCSE Virtual), 2024.
- [29] S. Yang, H. Zhao, Y. Xu, K. Brennan, and B. Schneider, "Debugging with an AI tutor: Investigating novice help-seeking behaviors and perceived learning," in Proc. ACM Conf. Int. Comput. Educ. Res. (ICER), Melbourne, Australia, 2024.
- [30] J. Leinonen et al., "Comparing code explanations created by students and large language models," in Proc. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE), Turku, Finland, 2023.
- [31] S. MacNeil et al., "Experiences from using code explanations generated by large language models in a web software development e-book," in Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE), Toronto, Canada, 2023.
- [32] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of AI code generators on supporting novice learners in introductory programming," in Proc. CHI Conf. Human Factors Comput. Syst., Hamburg, Germany, 2023.
- [33] M. Kazemitabaar, X. Hou, A. Z. Henley, B. J. Ericson, D. Weintrop, and T. Grossman, "How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment," in Proc. Koli Calling Int. Conf. Comput. Educ. Res., Koli, Finland, 2023.
- [34] M. Wermelinger, "Using GitHub Copilot to solve simple programming problems," in Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE), Toronto, Canada, 2023.
- [35] J. Prather et al., "'It's weird that it knows what I want': Usability and interactions with Copilot for novice programmers," ACM Trans. Comput.-Human Interact., vol. 31, no. 1, pp. 1–31, 2023.
- [36] J. Prather et al., "The widening gap: The benefits and harms of generative AI for novice programmers," in Proc. ACM Conf. Int. Comput. Educ. Res. (ICER), Melbourne, Australia, 2024.
- [37] Y. Shi, R. Schmucker, M. Chi, T. Barnes, and T. Price, "KC-Finder: Automated knowledge component discovery for programming problems," in Proc. 16th Int. Conf. Educ. Data Mining (EDM), 2023.
- [38] R. Niousha, M. Hoq, B. Akram, and N. Norouzi, "Use of large language models for extracting knowledge components in CS1 programming exercises," in Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE), 2024.
- [39] M. A. Demirtaş, M. Fowler, and K. Cunningham, "Reexamining learning curve analysis in programming education: The value of many small problems," in Proc. 17th Int. Conf. Educ. Data Mining (EDM), 2024.
- [40] I. Azaiz, N. Kiesler, and S. Strickroth, "Feedback-generation for programming exercises with GPT-4," in Proc. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE), Milan, Italy, 2024.
- [41] T. N. B. Duong and C. Y. Meng, "Automatic grading of short answers using large language models in software engineering courses," in Proc. IEEE Global Eng. Educ. Conf. (EDUCON), 2024.
- [42] E. Smolić, M. Pavelić, B. Boras, I. Mekterović, and T. Jaguš, "LLM generative AI and students' exam code evaluation: Qualitative and quantitative analysis," in Proc. 47th MIPRO ICT Electron. Conv., 2024.
- [43] P. Lagakis, S. Demetriadis, and G. Psathas, "Automated grading in coding exercises using large language models," in Smart Mobile Communication & Artificial Intelligence (IMCL 2023), LNNS, vol. 936. Cham, Switzerland: Springer, 2024, pp. 363–373.