
SOCRATICREPAIR: PLANNING-GUIDED SOCRATIC TUTORING FOR PROGRAM DEBUGGING AND MISCONCEPTION REPAIR

Mykhailo Kopytskyi
University of Warwick
mykhailo.kopytskyi@warwick.ac.uk

Mir Seyedebrahimi
Warwick Manufacturing Group
Mir.Seyedebrahimi@warwick.ac.uk

Jianhua Yang
Warwick Manufacturing Group
Jianhua.Yang@warwick.ac.uk

ABSTRACT

Large language models are increasingly used in computer science education. However, the teaching abilities of even the most powerful LLMs are limited: they tend to reveal the solution prematurely and may lose focus on the learner’s current debugging objective. To address this, we introduce SocraticRepair, a planning-guided Socratic tutoring framework for program debugging and misconception repair that adapts prior planning-guided tutoring work [Li et al., 2026b]. The framework is instantiated in the Python programming domain by fine-tuning Qwen2.5-Coder-7B-Instruct on synthetic multi-turn tutoring dialogues constructed from four preprocessed programming datasets. The system is evaluated using a predefined set of independent metrics and benchmarked against a general-purpose LLM. The experiments show that SocraticRepair achieves higher scores on stepwise scaffolding and question quality while also substantially reducing premature solution disclosure. The results also identify reliable pedagogical plan generation as an important remaining challenge.

1 Introduction

Large language models have demonstrated their usefulness in a variety of programming education scenarios, including program repair, hint generation, feedback and code explanation [Phung et al., 2023, Koutcheme et al., 2024]. Their ability to generate context-sensitive responses has further motivated their use in interactive educational settings [Kazemitabaar et al., 2024, Ma et al., 2024]. Consequently, there has been growing interest in using LLMs as conversational tutors that can guide learners through problem solving.

Designing and implementing effective tutoring remains one of the challenges. Existing systems have explored several approaches to make student-to-tutor interactions more pedagogically aligned. One line of work grounds general-purpose LLMs through carefully designed prompts, pedagogical guardrails and course-specific material with the aim of producing relevant assistance that avoids direct solution disclosure [Kazemitabaar et al., 2024, Ma et al., 2024]. A second line introduces more explicit structure into tutoring process by using multi-turn planning and knowledge tracing to determine how the interaction should proceed [Kargupta et al., 2024, Wang et al., 2025]. More recently, supervised fine-tuning has been explored as an alternative strategy to simpler prompting-based methods [Ross et al., 2025]. Despite the progress, prior work has mainly addressed these aspects of tutoring separately. In contrast, recent work outside programming education has demonstrated that a pedagogical tutor, grounded in a plan, can be incorporated into a specialised fine-tuned tutor [Li et al., 2026b]. This leaves it underexplored whether such structured tutoring behaviour can be adapted and learned by a specialised programming model for Socratic multi-turn bug and misconception repair, where the tutor must reason not only about the learner’s developing understanding, but

also about the current executable state of the learner’s program, whose correctness does not necessarily imply that the underlying misconception has been resolved.

To address the gap, we introduce SocraticRepair, a planning-guided tutoring framework for Socratic multi-turn programming bug and misconception repair. Building on recent work on planning-guided tutoring [Li et al., 2026b], the approach is adapted, such that the progress involves both the learner’s demonstrated understanding and the functional state of their code. SocraticRepair uses a pedagogical plan containing conceptual and repair objectives and conditions its guidance on the learner state and progress through these plan objectives. Because suitable multi-turn programming tutoring data are limited, a verified synthetic corpus is constructed from buggy student submissions drawn from four programming datasets. This corpus is used to fine-tune the Qwen2.5-Coder-7B-Instruct for planning and multi-turn tutoring. SocraticRepair is evaluated in controlled multi-turn interactions against a general-purpose LLM using 6 dimensional turn-level metrics and a binary metric of premature solution leakage.

The main contributions of this work are therefore threefold. **First**, a planning-guided multi-turn tutoring framework is adapted to Socratic multi-turn bug and misconception repair, where tutoring progress depends on both the learner’s demonstrated understanding and the functional state of their code. The framework is instantiated and evaluated in the Python programming domain. **Second**, a verified synthetic multi-turn tutoring corpus is constructed from four Python programming datasets using controlled student-to-tutor simulation and use it to fine-tune a Qwen2.5-Coder-7B-Instruct model for planning and tutoring. **Third**, the resulting tutor is evaluated under controlled interactions against a general-purpose LLM using six turn-level evaluation dimensions and a binary metric of premature solution leakage.

2 Literature Review

2.1 Intelligent Tutoring Systems (ITSs)

Before the emergence of LLM-based tutors, personalised computer-based tutoring was performed via the ITSs [VanLEHN, 2011, Kulik and Fletcher, 2016, Crow et al., 2018]. Traditional ITS architectures combined the knowledge about the target domain with a model of the learner and a pedagogical strategy for selecting the next action [Anderson et al., 1995]. Learner knowledge was represented in different ways across the systems: model tracing compared learner actions against a cognitive model, knowledge tracing estimated the probability that individual skills had been mastered from the observed learner performance while constraint-based modeling diagnosed errors through violations of domain constraints [Anderson et al., 1995, Corbett and Anderson, 1995, Ohlsson, 1994].

Programming tutors extended learner modeling with analysis of the learner’s program: early systems estimated mastery over individual programming rules and diagnosed errors from incorrect programs, while later ones used partial program states to generate adaptive feedback [Corbett and Anderson, 1995, Johnson, 1990, Rivers and Koedinger, 2017]. However, such systems generally relied on substantial domain-specific knowledge, making it hard to transfer to a new domain or adapt to an unseen programming problem.

2.2 Socratic Tutoring

Socratic tutoring treats questioning as a form of guided inquiry: rather than directly providing the answers, the tutor consistently investigates the learner’s assumptions and asks them to reconsider their reasoning [Overholser, 1993, Overholser and Beale, 2023]. Educational studies suggest that structured Socratic questioning can help critical thinking and conceptual understanding of the material, particularly when learners are required to actively respond to and reflect on the questions posed by the tutor [Yang et al., 2005, Knezic et al., 2010, Hsu et al., 2022].

Socratic tutors have also been used in the domain of programming education where they guide learners through the program code comprehension by questioning their interpretation of the program behaviour [Alshaikh et al., 2020, Tamang et al., 2021]. More recently, LLM-based approaches have extended this tutoring method to multi-turn debugging, investigating the effectiveness of both prompt-based and fine-tuned models as Socratic tutors [Al-Hossami et al., 2023, Kargupta et al., 2024, Al-Hossami and Bunescu, 2026].

However, they all differ in how the learner progress and tutoring objectives are represented and maintained across the conversation, leaving the integration of Socratic interaction with a learned tutoring model in the domain of bug and misconception repair relatively underexplored.

2.3 LLMs in Programming Education

Since the emergence of strong LLMs, they have been actively utilised in programming education. Prior work has explored their usefulness in a variety of scenarios, such as code generation, debugging, code explanation, feedback and hint generation [Phung et al., 2023, Sarsa et al., 2022, Raihan et al., 2025]. However, the tendency of general-purpose LLMs to provide direct assistance can conflict with the pedagogical objectives that require learners to reason through the problem themselves, while the provided assistance itself may not be related to what the learner would be expected to be taught to solve the programming exercise [Kazemitabaar et al., 2024, Kargupta et al., 2024, Zhao et al., 2026].

Several approaches have therefore adapted LLM assistance specifically for educational settings. Assignment-specific grounding has been used to keep responses consistent with the course material [Ma et al., 2024], while pedagogical prompting and guardrails have been used to limit direct solution leakage [Kazemitabaar et al., 2024].

Recent works on programming tutors introduce explicit mechanisms for managing the tutoring process through the conversation. TreeInstruct uses multi-turn planning and estimates of the learner’s knowledge to determine the next question [Kargupta et al., 2024]; TRAVER combines knowledge tracing with turn-level verification to guide learners toward completion of a coding task [Wang et al., 2025]. Misconception-oriented work structures debugging around reasoning trajectories intended to expose faulty beliefs about the program behaviour [Al-Hossami and Bunescu, 2026].

Outside of the domain of programming education, ScaffoldLM demonstrates that pedagogical planning, learner state assessment and progress tracking can be incorporated into a fine-tuned tutoring model for teaching mathematics [Li et al., 2026b]. However, the integration of planning-based Socratic tutor that supports bug and misconception repair into a learned tutoring model remains underexplored.

3 Methodology

3.1 SocraticRepair Framework

SocraticRepair is a planning-guided multi-turn tutoring framework for bug and misconception repair, instantiated for Python programming language. The framework is shown in Figure 1. We adopt the tutoring framework introduced by ScaffoldLM [Li et al., 2026b]. Their formulation is adapted to program and misconception repair by defining tutoring progress over both conceptual understanding and code repair by incorporating the state of the learner’s program as an additional signal.

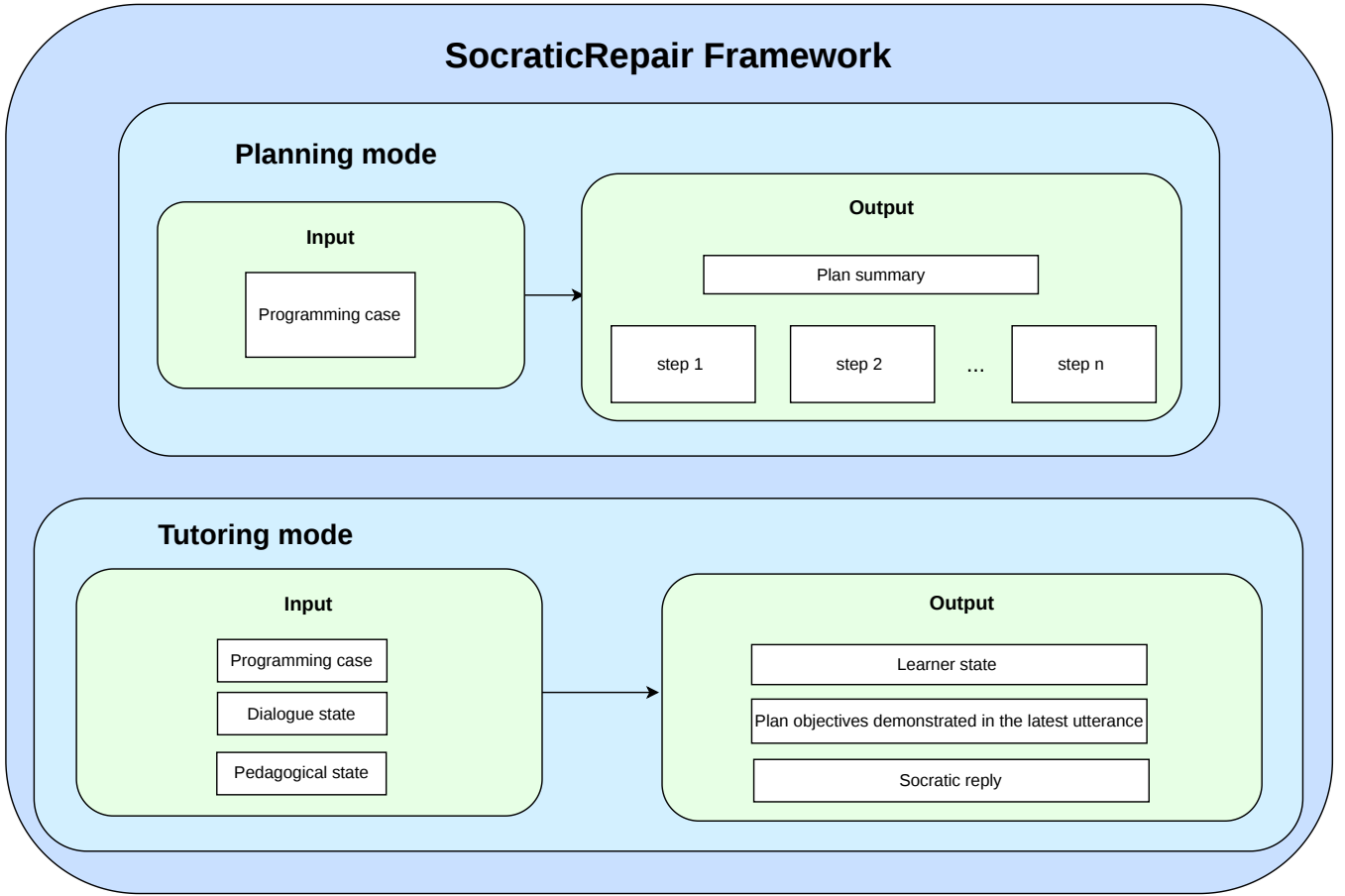


Figure 1: **SocraticRepair framework.** The same fine-tuned model operates in planning and tutoring modes. In planning mode, the model generates a stepwise pedagogical plan from the programming case. In tutoring mode, the model generates the learner state, step assessments (undemonstrated steps that have been demonstrated in the latest student utterance) and a Socratic reply given a programming case, latest submitted code, execution output, a plan and a progress through the plan.

The framework operates in two modes: planning and tutoring. In planning mode, the model receives the programming case together with the correct code solution and generates the pedagogical plan which is then used throughout the subsequent tutoring interaction. In tutoring mode, the model is conditioned on the programming case, pedagogical plan, current progress, dialogue history and the learner’s latest code and execution result. It jointly predicts the learner state, determines which previously unresolved plan objectives have been demonstrated or not demonstrated in the latest student utterance and generates the next Socratic response. Newly demonstrated plan objectives are then accumulated into the progress state supplied at the next tutor turn.

Progress is represented as a set of demonstrated objectives rather than in a strictly sequential manner, because some cases may involve several intertwined bugs, the repair of which can be performed in a different order by different students. The latest student utterance may therefore demonstrate any currently undemonstrated objective, including objectives scheduled later in the plan, while the first remaining undemonstrated objective is regarded as the current tutoring target.

The learner state characterises the latest student utterance relative to the current plan step and determines the tutor’s next pedagogical action. The completion of the plan step is done separately: a correct or comprehension state are only necessary but not sufficient for the demonstration of the current step, since sufficient evidence for the objective must also be present. Table 1 summarises the corresponding tutoring policy.

Learner state	Tutoring action	Active objective
START	Ask an initial guiding question	Unchanged
INCORRECT	Re-ask or provide a targeted hint	Not demonstrated
CONFUSION	Simplify the current objective or provide a hint	Not demonstrated
QUESTION	Answer the clarification and steer the student back to the objective	Not demonstrated
IRRELEVANT	Refocus the student on the current objective	Not demonstrated
CORRECT / COMPREHENSION	Advance when sufficient evidence demonstrates the objective, otherwise continue eliciting evidence	May be demonstrated

Table 1: Learner state tutoring policy, adapted from [Li et al., 2026b]. Unlike ScaffoldLM, SocraticRepair requires sufficient evidence that a pedagogical objective has been demonstrated before marking it as completed; the learner state alone is not sufficient.

Whenever the learner proposes modified code, it is executed against the available tests. Functional correctness is therefore treated separately from demonstrated conceptual progress: the program code may satisfy the tests while pedagogical plan objectives may still remain unresolved. The interaction therefore terminates only when all the plan steps have been marked as demonstrated and the student’s code passes all the tests.

3.2 Source Datasets and Preprocessing

The training and evaluation data are constructed from four Python programming datasets: PYMETA [Li et al., 2026a], MULTI_DEBUG [Kargupta et al., 2024], QUIXBUGS [Lin et al., 2017] and INTRO_PROG [Koutchme, 2023]. Together, these datasets provide variation in problem complexity and bug structure.

The datasets differ substantially in their original structure and available information and therefore require dataset-specific preprocessing and conversion to the common benchmark representation. Table 2 records the origin of each of the fields for all the source datasets. Cases for which the required information cannot be established reliably or the case itself is malformed are discarded.

Dataset	Case ID	Problem	Buggy code	Tests	Failure	Bugs description	Correct code	Source
PYMETA	D	G	D	G	X	G	G	D
MULTI_DEBUG	D	D	D	G	X	D	D	D
QUIXBUGS	D	D	D	D	X	G	D	D
INTRO_PROG	D	C	C	D	X	G	C	D

Table 2: Origin of the fields used in the common benchmark representation. C denotes a value copied directly from the source dataset; D denotes a value deterministically extracted; G denotes content generated or reconstructed by an LLM-based preprocessing agent; and X denotes a value obtained by executing the buggy implementation against the tests.

To reduce leakage between partitions, cases corresponding to the same underlying programming task are grouped before splitting. Candidate groups were constructed using the semantic similarity between programming tasks and the resulting grouping was manually inspected for consistency and potential overlap. Groups were then assigned as indivisible units to the training, validation and test sets. Table 3 shows the resulting distribution across the four data sources of 526 programming cases into the final train, validation and test sets.

3.3 Synthetic Dialogue Generation

Before dialogue generation, a pedagogical plan and student profile are generated for each programming case using GPT-5-mini and GPT-5-nano respectively. The plan is conditioned on the programming case and its bug and repair information (training only) and contains three to eight pedagogical objectives. A separate verifier agent, GPT-5.6-luna, checks the generated plan is consistent and technically correct, otherwise the plan is regenerated up to 3 times. The student profile is generated once per case and contains a fixed student background with one to five misconceptions

Dataset	Train	Validation	Test	Total
PYMETA	242	29	29	300
MULTI_DEBUG	120	15	15	150
QUIXBUGS	32	4	4	40
INTRO_PROG	26	5	5	36
Total	420	53	53	526

Table 3: Distribution of programming cases across data sources and train, validation and test partitions.

about their program. The student profile is present only during the dialogue generation step as it helps the student LLM keep their erroneous beliefs related to the bugs in the program. Both the plan and the student profile are reused across the three behavioural variants: receptive, uncertain and persistent.

Dialogue generation proceeds by first generating a student response. The student agent, GPT-5-mini, is provided with all the information about the programming case, including the correct code. For each programming case, 3 dialogues are generated, each with one of the three student variants. The behavioural variant determines a probability distribution over turn-level learner states. The first student turn uses the start state, whereas the subsequent turns are sampled from correct, incorrect, question, comprehension, confusion and irrelevant according to Table 4. This allows to simulate different student behaviours for the same programming problem. The student verifier agent, GPT-5.6-luna, checks that the student’s utterance is consistent with the sampled learner state, otherwise it is regenerated up to 3 times.

Learner state	RECEPTIVE	UNCERTAIN	PERSISTENT
CORRECT	0.40	0.30	0.25
INCORRECT	0.20	0.15	0.35
QUESTION	0.10	0.25	0.15
COMPREHENSION	0.10	0.10	0.05
CONFUSION	0.10	0.15	0.15
IRRELEVANT	0.10	0.05	0.05

Table 4: Learner state sampling probabilities for the three simulated student variants.

For each accepted student response, the tutor agent, GPT-5-mini, is provided with all the information about the programming case as well as the plan, current progress, latest submitted code and dialogue history to generate the next Socratic reply. If the student provides the new code, the tutor executes it and the output is included in the prompt. The tutor verifier agent checks that the tutor response is technically correct and aligns with the current active step and the whole dialogue overall, otherwise it is regenerated up to 3 times.

Dialogue generation continues until all the steps in the progress frontier are marked as completed and the latest student code passes all the tests. If the number of rounds exceeds 14 or the dialogue verifier agent, GPT-5.6-luna, spots an overall inconsistency of the dialogue, then it is regenerated up to 3 times.

3.4 Training Data Construction and Fine-tuning

A total of 1213 accepted dialogues were generated, covering 419 unique programming cases and containing 8493 tutor turns together with 419 pedagogical plans and 419 student profiles. The resulting dialogues and pedagogical plans are converted into two supervised tasks corresponding to two operating modes of SocraticRepair: pedagogical planning and tutoring. For the planning mode, each example contains the programming case and correct code solution as input, with the pedagogical plan and plan summary as the target. For the tutor mode, each tutor turn is converted into an individual example conditioned on the programming case, pedagogical plan, current progress, current program state and dialogue history. The target consists of the learner state, a list of demonstrated or not demonstrated decisions for every yet undemonstrated plan step and a Socratic reply. Inconsistent turn-level samples have been removed, reducing the 8493 generated tutor turns to final 8183 examples.

The two tasks are combined to fine-tune Qwen2.5-Coder-7B-Instruct using supervised fine-tuning with Low-Rank Adaptation (LoRA). The 419 plans are repeated twice during training and mixed with all 8183 tutor turns before shuffling. Training is performed for two epochs with LoRA rank 32, scaling factor 64 and a dropout 0.05 using a learning rate of 1×10^{-4} , an effective batch size of 8 and a maximum sequence length of 12288 tokens. Loss is computed over the assistant completion by masking all prompt tokens.

4 Evaluation

4.1 Evaluation Setup

We evaluate three tutor variants under the identical controlled student interactions. The first is the complete SocraticRepair pipeline, where the fine-tuned model generates both the pedagogical plan and tutor responses. The second uses the same SocraticRepair tutor with plans generated by the GPT-4o and verified by the GPT-5.6-luna. The third uses GPT-4o which was selected as the general-purpose LLM for Socratic tutoring. The tutors are evaluated through controlled multi-turn dialogues generated on the 53 held-out programming cases. For each programming case, the same student profile is reused across the three behavioural student variants, producing 159 dialogues for each tutor variant. Student responses are generated by GPT-5-mini and verified by the GPT-5.6-luna agent, with up to three generation attempts.

The generated dialogues are converted into individual tutor turns where each turn is evaluated in the context of the current programming case, dialogue history and the latest student response. Six turn-level dimensions are adapted from prior tutoring evaluations [Li et al., 2026b, Kargupta et al., 2024]. The metrics are summarised in Table 5. Premature solution disclosure is recorded separately as a binary turn-level measure, while code repair success and interaction length are reported at the dialogue level.

Metric	Evaluated property	Literature basis
Technical correctness	Technical accuracy and consistency with the program, execution evidence and trusted repair.	ScaffoldEval: Answer Accuracy
Debugging relevance	Alignment with the student’s current objective.	ScaffoldEval: Topic Adherence; TreeInstruct: Relevance
Stepwise scaffolding	Step by step guidance that leaves meaningful reasoning work to the student.	ScaffoldEval: Stepwise Scaffolding; TreeInstruct: Indirectness
Guidance progression	Productive movement towards unresolved objectives without unnecessary repetition.	ScaffoldEval: Guidance Quality; TreeInstruct: Logical Flow
Adaptive feedback	Responsiveness to the student’s latest utterance.	ScaffoldEval: Adaptive Feedback
Debugging question quality	Use of questions that elicit useful explanation	ScaffoldEval: Question Quality

Table 5: Turn level tutor quality dimensions adapted from prior tutoring evaluations [Li et al., 2026b, Kargupta et al., 2024].

4.2 Results

4.2.1 Dialogue outcomes and premature disclosure

SocraticRepair successfully completes 148 of the 159 interactions (93.1%), compared with 152 (95.6%) for GPT-4o. The 11 incomplete SocraticRepair dialogues consist of eight student generation protocol failures and three tutor output failures, while all seven incomplete GPT-4o dialogues result from student generation failures.

SocraticRepair interactions are longer, with a mean of 8.04 rounds compared with 4.99 for the GPT-4o. Premature answer disclosure occurs in 24 of 1,276 SocraticRepair Tutor turns (1.88%), compared with 59 of 794 GPT-4o turns (7.43%). The difference is also visible at the dialogue level, where disclosure occurs in 20 SocraticRepair dialogues (12.6%) and 53 GPT-4o dialogues (33.3%). This reduction is consistent with SocraticRepair’s explicit training for incremental scaffolding and avoidance of premature answer disclosure. Table 6 summarises the main results.

	SocraticRepair	GPT-4o
Evaluation dialogues	159	159
Successful dialogues	148 (93.1%)	152 (95.6%)
Mean rounds	8.04	4.99
Premature-disclosure turns	24 / 1,276 (1.88%)	59 / 794 (7.43%)
Dialogues with disclosure	20 / 159 (12.58%)	53 / 159 (33.33%)

Table 6: Dialogue level outcomes and premature solution disclosure for SocraticRepair and GPT-4o.

4.2.2 Planner mode quality

The learned planner produces a valid pedagogical plan for 52 of the 53 held out cases. Inspection against the trusted programming case information identified a technical error in 41 out of 52 plans, as summarised in Table 7. In comparison, all 53 plans generated by the GPT-4o are accepted by the GPT-5.6-luna verifier, with 50 accepted on the first attempt and three after one regeneration. The verified-plan SocraticRepair condition is therefore used to report tutor quality results independently of variation introduced by the plan generation.

Plan source	Attempts	Technically valid
SocraticRepair	53	11
GPT-4o + verifier	53	53

Table 7: Technical validity of SocraticRepair-generated pedagogical plans for the held-out evaluation cases.

4.2.3 Tutor quality

For comparison between tutors, turn-level scores are first averaged within each dialogue. Differences are tested using two-sided Wilcoxon signed-rank tests. Holm correction is applied across the six comparisons to control the family-wise error rate. Table 8 compares the verified-plan SocraticRepair tutor with GPT-4o tutor for the 158 matched dialogues. SocraticRepair achieves higher scores for stepwise scaffolding and debugging question quality, which is consistent with SocraticRepair’s fine-tuning objective, whereas GPT-4o obtains higher technical correctness and adaptive feedback scores, which may reflect GPT-4o’s stronger general purpose programming capabilities. The difference in scores for the debugging relevance and guidance progression are not statistically significant.

Metric	SocraticRepair	GPT-4o	Δ	Holm-adjusted p
Technical correctness	1.789	1.860	-0.071	.002
Debugging relevance	1.929	1.949	-0.021	.118
Stepwise scaffolding	1.876	1.789	+0.087	.006
Guidance progression	1.655	1.661	-0.006	.385
Adaptive feedback	1.792	1.843	-0.051	.036
Debugging question quality	1.626	1.446	+0.180	< .001

Table 8: Dialogue averaged tutor quality scores over 158 matched interactions. Scores range from 0 to 2 and Δ denotes SocraticRepair minus GPT-4o. Statistical significance is assessed using paired two-sided Wilcoxon signed-rank tests with Holm correction across the six metrics.

4.2.4 Adaptation across learner states

The adaptive feedback metric is at its most important when the student requires a change in the tutoring strategy. Therefore, this metric is examined separately across these learner states: incorrect, question, confusion and irrelevant in Table 9. Across those learner states, neither of the tutors shows a consistent advantage: GPT-4o scores slightly higher on responses following incorrect and question learner states, while SocraticRepair scores higher on responses following confusion and irrelevant. SocraticRepair predicts these four learner states with 91.9–100% accuracy. Although exact match accuracy for comprehension is only 58.1%, all 49 misclassifications are predictions of correct,

which is still a positive learner state.

Learner state	Turns (SR/GPT)	SR Adapt.	GPT Adapt.	SR state acc.
CORRECT	430 / 213	1.840	1.939	95.3%
INCORRECT	236 / 150	1.640	1.660	91.9%
QUESTION	156 / 103	1.833	1.864	100.0%
COMPREHENSION	117 / 56	1.897	1.982	58.1%
CONFUSION	109 / 67	1.560	1.522	98.2%
IRRELEVANT	69 / 47	1.812	1.809	100.0%

Table 9: Adaptive feedback scores by controlled learner state and SocraticRepair learner state prediction accuracy. The fixed START state is omitted.

4.2.5 Performance by bug count

Table 10 examines whether tutor behaviour changes with the number of bugs in a programming case. SocraticRepair’s advantage in debugging question quality is observed across all of the three groups and is the largest for cases containing three or more bugs. Its technical correctness remains below the GPT-4o one in each group. It can also be observed that even though GPT-4o wins overall on the guidance progression, SocraticRepair scores slightly higher on this metric for cases with three or more bugs, which is consistent with explicit pedagogical planning becoming more useful when several debugging objectives must be tracked.

Bugs	Matched	Tech.	Relevance	Scaff.	Progression	Adapt.	Q. qual.
1	74	-0.094	-0.004	+0.137	-0.015	-0.037	+0.157
2	42	-0.057	-0.045	+0.014	-0.017	-0.098	+0.113
3+	42	-0.044	-0.025	+0.072	+0.020	-0.027	+0.289

Table 10: Difference between SocraticRepair and GPT-4o by annotated bug count (Δ = SocraticRepair minus GPT-4o).

4.2.6 Judge agreement

To assess the reliability of the automatic evaluation, a subset of 40 tutor turns was randomly selected and independently scored by an undergraduate 3rd year Computer Science student using the same metrics. Agreement on the six dimensions is measured using linearly weighted Cohen’s κ , while standard Cohen’s κ is used for the binary premature disclosure metric. The results in Table 11 demonstrate a substantial agreement across the six tutor quality dimensions. Agreement is highest for the technical correctness, while the more subjective metric of adaptive feedback shows the lowest agreement of 0.66.

Evaluation dimension	Cohen’s κ
Technical correctness	0.81
Debugging relevance	0.77
Stepwise scaffolding	0.70
Guidance progression	0.68
Adaptive feedback	0.66
Debugging question quality	0.72
Premature disclosure	0.83

Table 11: Agreement between the human annotator and an LLM judge. Linearly weighted Cohen’s κ is reported for the six tutor quality dimensions, while standard Cohen’s κ is used for the binary premature disclosure metric.

5 Conclusion

This work has introduced SocraticRepair, a planning-guided tutoring framework for multi-turn bug and misconception repair adapted from [Li et al., 2026b]. The framework was instantiated and evaluated in the Python programming domain. The framework combines a pedagogical plan with learner state assessment, progress tracking and executable program feedback. The framework is implemented by fine-tuning Qwen2.5-Coder-7B-Instruct on verified synthetic dialogues constructed from four programming datasets.

The evaluation reveals a clear difference between the reliability of the two learned modes of SocraticRepair. While the tutoring mode demonstrates the intended pedagogical behaviour when conditioned on the verified plans, the learned planning mode frequently produces technically incorrect pedagogical plan steps. Under the verified-plan condition, SocraticRepair tutor mode achieves higher stepwise scaffolding and debugging question quality scores than the GPT-4o and substantially reduces the premature solution disclosure. GPT-4o remains stronger on technical correctness and adaptive feedback, although learner state conditioned analysis does not show that GPT-4o has a consistent advantage on the learner states where adaptation is most needed.

6 Limitations and Future Work

The study has several limitations. The **first** is that the model is fine-tuned on the synthetic data. While variability in learner behaviour was introduced during the synthetic dialogue generation, these interactions may still not reflect how real learners would interact with the tutor. **Secondly**, the scale of the synthetic corpus was constrained by the API cost of multi-agent generation, verification and repeated regeneration. Consequently, the model was trained on 1213 accepted dialogues covering 419 programming cases. **Thirdly**, the learned planning mode remains unreliable, with many generated plans containing technical errors. Future work should therefore investigate the use of a stronger base model with greater programming capability and whether planning should be separated from tutoring into a different agent. **Fourthly**, turn-level tutor quality is primarily assessed by an LLM judge. Although human annotation of a subset of turns shows a substantial agreement with the LLM evaluator agent, annotation of all the turn-level turns involving multiple postgraduate human raters would provide more accurate results.

References

- Erfan Al-Hossami and Razvan Bunescu. Reasoning trajectories for socratic debugging of student code: From misconceptions to contradictions and updated beliefs, 2026. URL <https://arxiv.org/abs/2511.00371>.
- Erfan Al-Hossami, Razvan Bunescu, Justin Smith, and Ryan Teehan. Can language models employ the socratic method? experiments with code debugging, 2023. URL <https://arxiv.org/abs/2310.03210>.
- Zeyad Alshaikh, Lasagn Tamang, and Vasile Rus. A socratic tutor for source code comprehension. In *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part II*, page 15–19, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-52239-1. doi:10.1007/978-3-030-52240-7_3. URL https://doi.org/10.1007/978-3-030-52240-7_3.
- John R. Anderson, Albert T. Corbett, Kenneth R. Koedinger, and Ray. Pelletier. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4(2):167–207, 1995. doi:10.1207/s15327809jls0402_2. URL https://doi.org/10.1207/s15327809jls0402_2.
- Albert T. Corbett and John R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1995. doi:10.1007/BF01099821.
- Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. Intelligent tutoring systems for programming education: a systematic review. In *Proceedings of the 20th Australasian Computing Education Conference, ACE ’18*, page 53–62, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450363402. doi:10.1145/3160489.3160492.

-
- Fu-Hui Hsu, I-Hsiu Lin, Hui-Chin Yeh, and Nian-Shing Chen. Effect of socratic reflection prompts via video-based learning system on elementary school students' critical thinking skills. *Computers & Education*, 183:104497, 2022. ISSN 0360-1315. doi:<https://doi.org/10.1016/j.compedu.2022.104497>. URL <https://www.sciencedirect.com/science/article/pii/S0360131522000689>.
- W.Lewis Johnson. Understanding and debugging novice programs. *Artificial Intelligence*, 42(1):51–97, 1990. ISSN 0004-3702. doi:[https://doi.org/10.1016/0004-3702\(90\)90094-G](https://doi.org/10.1016/0004-3702(90)90094-G). URL <https://www.sciencedirect.com/science/article/pii/000437029090094G>.
- Priyanka Kargupta, Ishika Agarwal, Dilek Hakkani-Tur, and Jiawei Han. Instruct, not assist: Llm-based multi-turn planning and hierarchical questioning for socratic code debugging, 2024. URL <https://arxiv.org/abs/2406.11709>.
- Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 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*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi:10.1145/3613904.3642773. URL <https://doi.org/10.1145/3613904.3642773>.
- Dubravka Knezic, Theo Wubbels, Ed Elbers, and Maaïke Hajer. The socratic dialogue and teacher education. *Teaching and Teacher Education*, 26(4):1104–1111, 2010. ISSN 0742-051X. doi:<https://doi.org/10.1016/j.tate.2009.11.006>. URL <https://www.sciencedirect.com/science/article/pii/S0742051X09002601>.
- Charles Koutchme. Introprog: A dataset of student submissions to introductory programming assignments. Hugging Face Datasets, 2023. URL https://huggingface.co/datasets/koutch/intro_prog.
- Charles Koutchme, Nicola Dainese, and Arto Hellas. Using program repair as a proxy for language models' feedback ability in programming education. In Ekaterina Kochmar, Marie Bexte, Jill Burstein, Andrea Horbach, Ronja Laarmann-Quante, Anaïs Tack, Victoria Yaneva, and Zheng Yuan, editors, *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 165–181, Mexico City, Mexico, June 2024. Association for Computational Linguistics. URL <https://aclanthology.org/2024.bea-1.15/>.
- James A. Kulik and J. D. Fletcher. Effectiveness of intelligent tutoring systems: A meta-analytic review. *Review of Educational Research*, 86(1):42–78, 2016. doi:10.3102/0034654315581420. URL <https://doi.org/10.3102/0034654315581420>.
- Chuyue Li, Ziqi Tang, Jingyi Wang, Yu Wu, Kazuma Hashimoto, and Lingyu Gao. Pymeta: A benchmark dataset for hierarchical student code error classification with python-interpreter-based labels. <https://huggingface.co/datasets/CircleCat/pymeta>, 2026a. CircleCat.
- Zechen Li, Qiannan Zhu, Mei Wang, Jia Li, and Hua Huang. Planning-guided tutoring with assessment-driven memory for pedagogical LLM tutors. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7165–7188, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi:10.18653/v1/2026.acl-long.325. URL <https://aclanthology.org/2026.acl-long.325/>.
- Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. Quixbugs: A multi-lingual program repair benchmark set based on the quixey challenge. In *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, pages 55–56. ACM, 2017. doi:10.1145/3135932.3135941. URL <https://github.com/jkoppel/QuixBugs>.
- Iris Ma, Alberto Krone Martins, and Cristina Videira Lopes. Integrating ai tutors in a programming course, 2024. URL <https://arxiv.org/abs/2407.15718>.

-
- Stellan Ohlsson. Constraint-based student modeling. In Jim E. Greer and Gordon I. McCalla, editors, *Student Modelling: The Key to Individualized Knowledge-Based Instruction*, pages 167–189, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. ISBN 978-3-662-03037-0.
- James C. Overholser. Elements of the socratic method: I. systematic questioning. *Psychotherapy*, 30:67–74, 1993. URL <https://api.semanticscholar.org/CorpusID:52885059>.
- James C. Overholser and Eleanor Beale. The art and science behind socratic questioning and guided discovery: a research review. *Psychotherapy Research*, 33(7):946–956, 2023. doi:10.1080/10503307.2023.2183154. URL <https://doi.org/10.1080/10503307.2023.2183154>. PMID: 36878221.
- Tung Phung, Victor-Alexandru Pădurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. Generative ai for programming education: Benchmarking chatgpt, gpt-4, and human tutors, 2023. URL <https://arxiv.org/abs/2306.17156>.
- Nishat Raihan, Mohammed Latif Siddiq, Joanna C.S. Santos, and Marcos Zampieri. Large language models in computer science education: A systematic literature review. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSETS 2025, page 938–944, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400705311. doi:10.1145/3641554.3701863. URL <https://doi.org/10.1145/3641554.3701863>.
- Kelly Rivers and Kenneth R. Koedinger. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64, 2017. ISSN 1560-4292. doi:<https://doi.org/10.1007/s40593-015-0070-z>. URL <https://www.sciencedirect.com/science/article/pii/S1560429226004488>.
- Emily Ross, Yuval Kansal, Jake Renzella, Alexandra Vassar, and Andrew Taylor. Supervised fine-tuning llms to behave as pedagogical agents in programming education, 2025. URL <https://arxiv.org/abs/2502.20527>.
- Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, ICER ’22, page 27–43, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391948. doi:10.1145/3501385.3543957. URL <https://doi.org/10.1145/3501385.3543957>.
- Lasang Jimba Tamang, Zeyad Alshaikh, Nisrine Ait Khayi, Priti Oli, and Vasile Rus. A comparative study of free self-explanations and socratic tutoring explanations for source code comprehension. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE ’21, page 219–225, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380621. doi:10.1145/3408877.3432423. URL <https://doi.org/10.1145/3408877.3432423>.
- KURT VanLEHN. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011. doi:10.1080/00461520.2011.611369. URL <https://doi.org/10.1080/00461520.2011.611369>.
- Jian Wang, Yinpei Dai, Yichi Zhang, Ziqiao Ma, Wenjie Li, and Joyce Chai. Training turn-by-turn verifiers for dialogue tutoring agents: The curious case of llms as your coding tutors, 2025. URL <https://arxiv.org/abs/2502.13311>.
- Ya-Ting C. Yang, Timothy J. Newby, and Robert L. Bill. Using socratic questioning to promote critical thinking skills through asynchronous discussion forums in distance learning environments. *American Journal of Distance Education*, 19(3):163–181, 2005. doi:10.1207/s15389286ajde1903_4. URL https://doi.org/10.1207/s15389286ajde1903_4.
- Jin Zhao, Marta Knežević, and Tanja Käser. Evaluating answer leakage robustness of llm tutors against adversarial student attacks, 2026. URL <https://arxiv.org/abs/2604.18660>.