From Correctness to Expert-Refined Misconception Signals with Human-in-the-Loop Validation in CS1 Programming Education

Abstract

Knowledge tracing in introductory programming is commonly framed as predicting whether a student will eventually solve the next problem. While useful, this approach reduces rich programming activity to correctness labels and offers limited support for instructional interpretation. This study examines whether learner modelling in CS1 can be improved by moving from correctness histories to expert-refined misconception signals derived from programming traces. Using CodeWorkout trace data, we compared three models on a held-out later-course split: a correctness-only model, a process-aware model using compilation, execution, and revision features, and a misconception-signal model that added expert-refined recurrent code-pattern signals. Because eventual correctness was highly imbalanced, we evaluated performance using macro F_1, AUC, and Brier score rather than relying on accuracy alone. The process-aware model achieved the strongest macro F_1, while the misconception-signal model achieved the best AUC and Brier score, indicating stronger discrimination and calibration than correctness-only histories. To examine pedagogical value, we conducted a human-in-the-loop evaluation. Experts judged 90.0% of candidate signals plausible and 90.0% actionable for teaching, and misconception-aware summaries were preferred in 90.0% of paired instructor evaluations. These summaries were also rated higher for credibility, usefulness, clarity, and actionability. The study shows that richer learner representations can move CS1 knowledge tracing beyond predicting outcomes alone towards explaining the nature of student struggle in ways instructors can interpret and use.

CCS Concepts

• **Social and professional topics → Computer science education**; *CS1*; Student assessment.

Keywords

knowledge tracing, CS1, programming education, misconception signals, human-in-the-loop, educational data mining

ACM Reference Format:

. 2026. From Correctness to Expert-Refined Misconception Signals with Human-in-the-Loop Validation in CS1 Programming Education. In . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Student modelling in introductory programming is often framed as a prediction problem. Given a learner's prior attempts, can we predict whether the next submission will compile, pass tests, or eventually be corrected? This line of work has produced influential models of temporal learning, beginning with Bayesian Knowledge Tracing and extending to neural approaches that learn from long sequences of learner interactions [6, 13]. Yet correctness is an incomplete account of learning in CS1. Two students may both fail on the same exercise while exhibiting different misunderstandings, and those differences matter for feedback, assessment, and instructional decision making. This limitation is especially important in programming, where student responses are not merely right or wrong answers but evolving code artefacts. A submission may reveal uncertainty about control flow, indexing, method invocation, state changes, or scope long before it reveals eventual success or failure. Prior work has therefore moved beyond binary correctness towards richer representations of learner performance [11, 19]. Option Tracing showed that modelling the exact selected response can reveal patterns hidden by correctness labels alone [8]. Open-ended Knowledge Tracing made a comparable move in computing education by treating student code as the response to be modelled rather than collapsing it into a score [10]. These studies suggest that if the goal is to support teaching, student models should preserve diagnostic information rather than discard it.

A natural next step is to model misconceptions over time. In computing education, misconceptions have long been recognised as central to novice performance [3, 15]. Studies of exam scripts, classroom data, and large-scale submission traces have shown that novice errors are often systematic rather than random, and that many of them reflect recurring misunderstandings about core programming concepts [2, 4, 20]. More recent work has explored semi-automatic misconception discovery from code embeddings [18] and has begun to formulate misconception mining from student code as a benchmark task in its own right [1]. This body of work makes a strong case that misconceptions should be studied directly. However, it does not by itself solve the problem of temporal learner modelling in CS1. Knowledge tracing research has typically focused on temporal prediction [12, 17], while misconception research has typically focused on identifying, grouping, or describing error patterns [3, 15]. The two strands have not yet been integrated in a way that is both instructionally credible and empirically testable for CS1 programming. A further difficulty is that misconception resources are often language- and context-specific [5, 16]. A misconception inventory developed for one programming language cannot safely be transferred unchanged to another. For this reason, misconception-aware modelling in CS1 requires more than attaching an existing bank of labels to a sequence dataset. It requires a process for inferring candidate signals from authentic student traces and then testing whether those signals are predictive, interpretable, and educationally useful.

This need aligns well with the kinds of data now available in programming education. CodeWorkout was designed to support short programming exercises with fine-grained data collection [7], and ProgSnap2 has provided a common format for programming process data that captures successive code states and event histories [14]. Such datasets allow researchers to observe not only whether students eventually solve a problem, but also how they reach that point, how many attempts they require, what kinds of intermediate failures occur, and whether those failures recur across tasks. At the same time, the broader computing education literature has argued that programming process data can support more instructionally useful forms of educational data mining than final outcomes alone [9].

In this paper, we propose a knowledge tracing approach that moves from correctness labels to expert-refined misconception signals. Rather than assuming that misconceptions are fully known in advance, we treat them as candidate constructs inferred from recurrent code and process patterns and then refined through expert review. The study makes three contributions: it introduces a pipeline for inducing candidate misconception signals from programming traces and refining them through expert review; it evaluates whether those signals improve knowledge tracing beyond correctness-only and process-aware baselines; and it tests whether misconception-aware outputs are judged by instructors to be more credible and more useful for teaching than correctness-only summaries.

The study is guided by three research questions:

RQ1. To what extent are expert-refined misconception signals inferred from CS1 code submissions and process traces judged plausible and instructionally useful?

RQ2. Do misconception-signal knowledge tracing models improve prediction of future outcomes over correctness-only baselines?

RQ3. Do misconception-aware outputs provide explanations that instructors judge to be more useful and more credible than correctness-only summaries?

By linking learner modelling with expert refinement and instructor judgement, this study seeks to move CS1 knowledge tracing beyond forecasting outcomes alone. Its central contribution is to show that learner models can be designed not only to predict whether students will struggle, but also to represent the nature of that struggle in a form that instructors can interpret and use. In this sense, the paper advances knowledge tracing in computing education from a narrow prediction task towards a more pedagogically grounded account of learner difficulty.

2 Methodology

2.1 Design

We used a mixed-methods design combining temporal modelling of student programming traces with human-in-the-loop expert validation. The study asked whether learner models built from expert-refined misconception signals outperform correctness-only baselines, and whether the resulting outputs are judged by instructors to be more credible and more useful for CS1 teaching.

The design had three stages. First, we reconstructed student problem-solving trajectories from fine-grained CodeWorkout logs [7, 14]. Second, we induced candidate misconception signals from recurrent patterns in code states, compiler outcomes, and revision behaviour, then refined those signals through structured expert review. Third, we compared correctness-only, process-aware, and misconception-aware knowledge tracing models on held-out later coursework, followed by a blinded instructor evaluation of model outputs. This design connects knowledge tracing [6, 13], response-level learner modelling [8, 10], programming process analytics [9], and prior work on novice misconceptions [1, 2, 4, 18].

2.2 Dataset and representation

The study used the CodeWorkout trace data from a single CS1 course [7]. The dataset contains 506 students, 50 programming problems across five assignments, 125,578 code states, and 360,176 logged events. Each event record includes an anonymised student identifier, timestamp, assignment and problem identifiers, event type, score, compile outcome, compiler message fields, and a link to the submitted code state. Student-level course grades are available separately, and the provided `early.csv` and `late.csv` files define an early-to-late split for development and held-out evaluation.

The primary unit of analysis was the problem attempt trajectory for a student on a given problem. For each student-problem pair, we ordered all associated code states and events chronologically and summarised them using three feature groups: outcome features, code-change features, and process features. Outcome features included compile success and run score where available. Code-change features captured structural edits between consecutive submissions, such as changes to operators, conditions, returns, and branch structure. Process features captured attempt count, time gaps, repeated submission of near-identical code, and local revision churn. This representation preserves the intermediate structure of programming activity rather than collapsing each problem to a single correct or incorrect label.

2.3 Candidate misconception signals

Because the trace data are Java CS1 submissions and misconception labels are not directly provided, we did not assume a fixed taxonomy in advance. Instead, we induced candidate misconception signals from recurrent trace patterns. This was necessary because misconception categories are often language- and context-specific [1, 20]. Initial signal induction was carried out by two researchers using a manual procedure over recurrent local failure patterns derived from code-structure, compiler, and process features. Patterns were grouped within and across problems to distinguish one-off dead ends from recurring forms of misunderstanding. A candidate signal was retained only if it recurred across multiple students, reappeared for the same student, or appeared across more than one problem. Disagreements about signal boundaries or labels were first addressed through consensus discussion and, where needed, resolved by third-researcher adjudication, producing an initial inventory of candidate signals for later expert refinement.

Each candidate signal was represented by a compact template consisting of a structural signature, a typical outcome profile, and representative code exemplars. In practice, these templates captured patterns such as unstable boundary conditions, malformed branching logic, inconsistent return behaviour, or repeated unsuccessful edits around a single construct. Signal assignment to student attempts was rule-based: a signal was attached when a submission matched the corresponding structural signature together with its associated trace features. The same assignment procedure was applied to held-out data, so the misconception-aware model used reproducible signal definitions rather than post hoc qualitative labels.

2.4 Human-in-the-loop refinement

Candidate signals were refined through structured expert review by CS1 educators with experience in teaching introductory programming and familiarity with common novice difficulties. Each signal was presented as an anonymised review card containing a short description, representative code states, and a compact summary of the associated trajectory. Experts judged whether the pattern reflected a plausible misconception, a transient error, or an underspecified pattern. For patterns judged plausible, they assigned or revised a label and grouped it into a broader concept family, such as conditional logic, boundary reasoning, method invocation, or return semantics. This stage served two purposes. First, it improved interpretability by turning recurrent trace patterns into human-readable learner-state indicators. Second, it acted as a validity check, since prior work has shown that educator beliefs and student data do not always align cleanly [4]. The expert review was applied to 30 candidate signals taken forward from the induction stage. This number was chosen as a practical compromise between coverage and review burden for a short conference study: it was large enough to include the most recurrent and instructionally salient patterns while still allowing detailed review of each signal.

Candidate signals were reviewed independently by three CS1 educators. We report aggregated agreement statistics in future work; in this short paper, we focus on the refinement outcomes and the final accepted, merged, and dropped signal categories. Experts judged each candidate signal against three criteria: whether it represented a plausible misconception rather than a transient slip, typographical error, or gaming strategy; whether it was instructionally actionable; and whether it should be retained, merged with a broader category, or discarded. Signals were merged when different surface patterns were judged to reflect the same underlying misunderstanding, and were dropped when they were better interpreted as stylistic issues, accidental syntax slips, or behaviour aimed at passing tests rather than evidencing conceptual difficulty. The output of this stage was an expert-refined inventory of misconception signals tailored to the CS1 trace data.

2.5 Models and prediction tasks

We compared three model families under the same supervised prediction setting, differing only in the information used to represent learner state. All three models were implemented using logistic regression classifiers so that differences in performance could be attributed to differences in representation rather than to different learning algorithms.

The correctness-only model used prior problem outcomes, problem identifiers, and attempt counts. This represented the standard view that learner state can be approximated from correctness histories alone [6, 13]. The process-aware model augmented this baseline with trace-derived process features, including compile success rates, score trajectories, time gaps, repeated failures, and code churn. The misconception-aware model added the expert-refined misconception signals to the same feature backbone, so that each student history was represented not only by correctness and process features, but also by the activation and persistence of specific misconception signals over time. All three models were trained using the early split and evaluated on the held-out late split. Validation during development was grouped by student to avoid leakage across folds. This design ensured that differences in performance could be attributed to differences in learner representation rather than differences in training data or evaluation procedure. The primary prediction target was `CorrectEventually` in the held-out later split. We also report results for the provided `Label` field as an auxiliary benchmark outcome, while avoiding stronger pedagogical interpretation because its semantics are not explicit in the dataset.

2.6 Evaluation

Model development used the early split, and final evaluation used the late split. This design reflects an authentic instructional setting in which earlier coursework is used to model later performance. To avoid leakage, validation during development was grouped by student. We report accuracy, macro F_1, AUC, and Brier score. Accuracy and macro F_1 capture predictive performance, while AUC and Brier score assess ranking and calibration. Improvements were tested with student-level paired bootstrap intervals.

To evaluate interpretability and instructional value, we conducted a blinded instructor study. For a stratified sample of learner histories, we generated paired summaries from the correctness-only and misconception-aware models. Instructors rated each summary for credibility, usefulness for instructional decision making, and clarity, and then indicated which summary they preferred. This allowed us to test not only whether misconception-aware modelling improved prediction, but also whether it produced outputs that instructors considered more educationally meaningful.

2.7 Contribution of the method

The method is distinctive in two respects. First, it treats within-problem revision behaviour as part of learner state rather than reducing each problem to a final outcome. Second, it embeds expert review within the construction of the misconception representation itself, so that the learner model is evaluated both by predictive performance and by pedagogical credibility.

3 Results

The trace data comprised 506 students, 50 programming problems across five assignments, 125,578 code states, and 360,176 logged events. The early split contained 14,317 student-problem records over 30 problems, while the held-out late split contained 9,386 records over 20 problems. The primary outcome, `CorrectEventually`, was highly imbalanced in both splits, with 93.9% positive cases in the early split and 94.3% in the late split. Mean attempts fell from 5.71 in the early split to 4.17 in the late split. This class imbalance makes raw accuracy difficult to interpret in isolation and motivates the use of macro F_1, AUC, and Brier score alongside accuracy.

3.1 Predictive performance

Table 1 reports predictive performance on the held-out late split for `CorrectEventually`. Because `CorrectEventually` is highly imbalanced in the held-out late split, accuracy is a potentially misleading metric. A model can achieve strong raw accuracy by predicting the majority class while still offering little value for distinguishing which students are more or less likely to succeed. For that reason, we place particular emphasis on macro F_1, AUC, and Brier score. Macro F_1 is more sensitive to minority-case performance than accuracy, AUC captures ranking quality independently of a fixed decision threshold, and the Brier score evaluates probabilistic calibration. Together, these measures provide a more appropriate basis for comparison under strong class imbalance.

This is evident in the majority baseline, which achieved the highest raw accuracy at 94.27% despite an AUC of only 0.50 and a macro F_1 of 48.52. In other words, a model can appear strong on accuracy while offering little ability to distinguish between more and less likely future success. The correctness-only model improved substantially on this baseline, achieving a macro F_1 of 61.60 and an AUC of 0.690. This indicates that prior correctness histories capture useful information about student state, but also that correctness alone provides only a limited view of how students progress through CS1 problems.

Adding compilation, execution, and revision features produced further gains. The process-aware model achieved the highest macro F_1 at 62.01 and increased AUC to 0.806, showing that programming-process traces contain substantial predictive value beyond correctness histories alone. The misconception-signal model achieved the highest AUC at 0.822 and the best Brier score at 0.0496, indicating the strongest discrimination and calibration among the evaluated approaches. Although its macro F_1 was slightly below the process-aware model, this result is important for interpretation. The process-aware model appears strongest for threshold-based classification, whereas the misconception-signal model is strongest for ranking and calibration. We therefore do not argue that misconception signals dominate every alternative on every metric. Rather, we argue that they change the character of the learner model: instead of only improving classification, they provide a more educationally meaningful representation of student difficulty while retaining competitive predictive performance.

Table 1: Predictive performance on the held-out late split for `CorrectEventually`.

Model	Accuracy	Macro F_1	AUC	Brier
Majority baseline	94.27	48.52	0.500	0.0540
Correctness-only KT	93.48	61.60	0.690	0.0684
Process-aware KT	93.85	62.01	0.806	0.0549
Misconception-signal KT	94.13	60.26	0.822	0.0496

3.2 Human-in-the-loop refinement and evaluation

The expert review stage evaluated 30 candidate misconception signals derived from the trace data. Of these, 27 (90.0%) were judged plausible misconceptions and 27 (90.0%) were judged actionable for teaching, with a mean confidence rating of 4.40 out of 5. After refinement, 25 signals were accepted, 2 were merged into broader categories, and 3 were dropped. The dropped cases were informative: hardcoding visible test cases was judged to reflect gaming rather than misunderstanding, using & in place of && was treated as likely typographical rather than conceptual, and redundant boolean returns were judged stylistic rather than misconception-based. Two boundary-related signals were merged into a broader off-by-one category. These outcomes show that the expert stage did not merely confirm the induced signals, but materially improved their interpretability and construct validity.

The instructor evaluation compared correctness-only and misconception-signal summaries for 30 learner-history cases. The misconception-signal summary was preferred in 27 of 30 cases (90.0%), while the correctness-only summary was preferred in only 1 case (3.3%); 2 cases (6.7%) were judged ties. Experts also indicated that they would use the misconception-signal summary in teaching in 27 of 30 cases (90.0%). One case was marked as suitable to use alongside the alternative summary, and 2 cases were judged unhelpful in either form.

Table 2 reports the summary ratings. Misconception-signal summaries received higher average ratings for credibility, usefulness, clarity, and actionability. The largest gap was in actionability, with a mean advantage of 2.43 points on the 5-point scale. This suggests that the main contribution of the richer learner-state representation is not only predictive but pedagogical: instructors found the misconception-aware summaries substantially more useful for deciding what feedback or explanation to provide. The qualitative comments help explain this result. In several cases, instructors preferred the misconception-aware summary because it moved from reporting an error to explaining the underlying difficulty. Typical comments included, *‘B gives the misconception, A just gives the error”*, *‘Knowing it’s an initialisation error saves me reading the trace”*, and *“B is highly actionable for feedback”*.

> **Example of a misconception-aware summary.** A learner showed repeated evidence of an *uninitialised-variable* pattern across recent submissions. A correctness-only summary could report only that the learner was likely to continue struggling on the next task. By contrast, the misconception-aware summary identified the likely source of difficulty and pointed to a more specific instructional response. Rather than treating the case as generic failure, an instructor could revisit variable initialisation and scope, and provide a short debugging example showing how values are introduced before use.

3.3 Answering the research questions

Collectively, the results support a two-part conclusion. First, programming process traces improve learner modelling beyond correctness histories alone, and expert-refined misconception signals

Table 2: Human-in-the-loop results from 30 paired instructor evaluations.

Measure	Correctness-only	Misconception-signal
Credibility	3.70	4.73
Usefulness	2.57	4.43
Clarity	4.03	4.80
Actionability	1.90	4.33
Preferred summaries (%)	3.3	90.0
Would use in teaching (%)	0.0	90.0

further improve ranking and calibration. Second, the misconception-signal representation is far more useful to instructors than correctness-only summaries. The human-in-the-loop stage, therefore, strengthens the approach in two ways: it filters and refines the induced signal inventory, and it shows that the richer learner-state representation has clear pedagogical value for CS1 teaching. The findings provide evidence for all three research questions, although the strength of that evidence differs by question.

RQ1. To what extent are expert-refined misconception signals inferred from CS1 code submissions and process traces judged plausible and instructionally useful? The results support this question. Thirty candidate signals were reviewed by experts, of which 27 (90.0%) were judged plausible misconceptions and 27 (90.0%) were judged actionable for teaching, with a mean confidence rating of 4.40 out of 5. After refinement, 25 signals were accepted, 2 were merged into broader categories, and 3 were dropped. This shows that candidate signals can be induced from the trace data and refined into a more credible instructional representation. The merge and drop decisions are particularly important because they show that the human-in-the-loop stage did not simply confirm all induced patterns, but helped distinguish pedagogically meaningful misconceptions from gaming, typographical slips, and stylistic issues.

RQ2. Do misconception-signal knowledge tracing models improve prediction of future outcomes over correctness-only baselines? The results also support this question, with some nuance. Compared with the correctness-only model, the misconception-signal model achieved higher accuracy (94.13 vs. 93.48), substantially higher AUC (0.822 vs. 0.690), and a better Brier score (0.0496 vs. 0.0684). These gains indicate improved discrimination and calibration over correctness-only histories. However, the misconception-signal model did not outperform every alternative on every metric, as the process-aware model achieved the highest macro F_1 (62.01). The evidence therefore suggests that misconception signals improve prediction over correctness-only baselines, particularly in ranking and calibration, while process features remain highly valuable for threshold-based classification.

RQ3. Do misconception-aware outputs provide explanations that instructors judge to be more useful and more credible than correctness-only summaries? This question received the strongest support. In paired instructor evaluations of 30 learner-history summaries, the misconception-aware summary was preferred in 27 cases (90.0%), while the correctness-only summary was preferred in only 1 case (3.3%); 2 cases (6.7%) were ties. Misconception-aware summaries also received higher average ratings for credibility (4.73 vs. 3.70), usefulness (4.43 vs. 2.57), clarity (4.80 vs. 4.03), and actionability (4.33 vs. 1.90). In addition, instructors indicated that they would use the misconception-aware summary in teaching in 27 of the 30 cases (90.0%), while no case selected the correctness-only summary alone for teaching use. These results show that the richer learner-state representation is not only predictive, but also substantially more educationally useful.

Taken together, the results show that expert-refined misconception signals can be inferred from CS1 traces, that they improve learner modelling beyond correctness-only histories in key predictive respects, and that they produce outputs instructors judge to be markedly more useful and more credible for instructional decision making.

Taken together, these findings show that the study contributes more than a modest improvement in prediction: it demonstrates that knowledge tracing in CS1 can be reoriented towards explaining the structure of learner difficulty in ways that instructors find credible and useful.

4 Discussion

This study examined whether CS1 learner modelling benefits from moving beyond correctness labels towards expert-refined misconception signals. The results suggest that it does, but in a specific way. The main gains did not appear as a simple increase in raw accuracy, which is difficult to interpret under the strong class imbalance in `CorrectEventually`. Instead, the richer representations were most valuable in discrimination, calibration, and interpretability. Process-aware modelling improved prediction beyond correctness histories, while the misconception-signal model achieved the strongest AUC and Brier score. More importantly, the human-in-the-loop evaluation showed that instructors overwhelmingly preferred the misconception-aware summaries. These findings suggest that misconception-aware knowledge tracing matters not only for forecasting outcomes, but for producing learner-state representations that better support teaching.

The results also clarify the relationship between process data and misconception modelling. Prior work in programming education has argued that process traces provide richer evidence of learning than final outcomes alone [9, 14]. Our findings support that claim. The process-aware model achieved the highest macro F_1, indicating that submission traces capture useful information about student difficulty beyond correctness histories. However, process traces alone do not guarantee pedagogical clarity. The misconception-signal model did not dominate on every metric, but it produced the best discrimination and calibration and the most instructionally valued summaries. This suggests that misconception signals matter because they organise process evidence into forms that teachers can recognise and use. A correctness-only model can indicate likely failure, but it cannot distinguish that generic risk from a specific pattern of difficulty. By contrast, a misconception-aware summary points an instructor towards a more targeted response, such as revisiting control flow, variable initialisation, or boundary reasoning. A second contribution is methodological. Existing work on novice programming misconceptions has shown that recurring error patterns can be identified in student data [2, 4, 18], while knowledge tracing research has shown how learner histories can be used for prediction [6, 13]. Our study connects these strands by embedding expert review within the construction of the learner representation. This matters because not every recurrent trace pattern should be treated as a misconception. In our expert review, several candidate signals were merged or dropped because they reflected gaming, typographical slips, or stylistic choices rather than substantive misunderstanding. One useful example is *hard-coding test cases*. A correctness-only model would treat such a case simply as success or failure, and an automatically induced signal might misleadingly present it as a misconception. The expert stage instead rejected it as gaming behaviour rather than conceptual difficulty. This strengthens the contribution by showing that human judgement is necessary for distinguishing educationally meaningful constructs from superficially frequent patterns.

The findings have direct implications for teaching. Learner analytics in CS1 should be judged not only by predictive performance but also by pedagogical usefulness. A model that predicts well but produces opaque explanations is of limited value in classroom settings. Our results suggest that misconception-aware representations may offer a practical bridge between learning analytics and instructional action. Instructors strongly preferred summaries that named recurring learner difficulties because they moved from reporting an error to explaining the nature of the struggle. This means that the system can support different interventions. A student associated with a boundary-related loop signal may need targeted work on indexing and inclusive versus exclusive conditions, whereas a student associated with *print vs return* may need support in distinguishing console output from method behaviour.

The study also has important limitations. It uses trace data from a single CS1 course and a single programming language, so the resulting signal inventory should not be treated as a universal taxonomy of novice programming difficulty. The dataset does not include full natural-language problem statements, which limits our ability to connect inferred signals to prompt semantics. In addition, the human-in-the-loop stage involved a relatively small number of expert judgements, which is appropriate for a short conference study but not sufficient to claim a definitive instructional ontology. The primary outcome was eventual correctness, which is highly imbalanced. For that reason, the strongest evidence comes from the combined pattern across discrimination, calibration, and instructor judgement rather than from accuracy alone. Some accepted signals are closely tied to Java semantics. However, the framework itself is not Java-specific and can be adapted to other introductory languages, including Python. In such cases, the transferable contribution is the methodology rather than the exact final inventory of signals.

Future work should extend the method across courses, languages, and intervention settings by testing whether the same human-in-the-loop pipeline supports adaptive feedback or problem selection. Overall, the study shows that moving from correctness to expert-refined misconception signals changes what a knowledge tracing model can contribute in CS1. The main advance is not that the richer model always predicts more accurately in a narrow statistical sense. It is that it produces a more educationally useful account of learner state, one that combines temporal evidence from programming traces with expert judgement about which recurring patterns are worth treating as misconceptions.

5 Conclusion

This paper examined whether CS1 learner modelling can be strengthened by moving beyond correctness histories towards expert-refined misconception signals derived from programming traces. Using CodeWorkout data, we compared correctness-only, process-aware, and misconception-signal knowledge tracing models on a held-out later-course split, alongside a human-in-the-loop evaluation.

The findings show a clear pattern. Correctness histories alone provide a useful but limited representation of learner state. Adding process features from compilation, execution, and revision traces improves predictive performance, while adding expert-refined misconception signals further improves discrimination and calibration, yielding the strongest AUC and Brier score among the evaluated models. More importantly, the human-in-the-loop results show that the richer learner representation is substantially more useful to instructors: misconception-aware summaries were overwhelmingly preferred and received higher ratings for credibility, usefulness, clarity, and actionability.

The main contribution of the study is therefore not simply a better predictive model. Its larger contribution is to move knowledge tracing in CS1 from predicting failure towards representing the nature of student struggle. By combining fine-grained programming traces with expert refinement, the approach produces learner-state representations that are both empirically informative and instructionally meaningful. In this sense, the study shows that learner models for programming education should be judged not only by whether they predict future success, but also by whether they generate explanations that educators can trust and use. Although the work is limited to one CS1 setting and one programming language, it offers a transferable framework for combining temporal prediction with pedagogical interpretation. Future work can extend this approach across courses, languages, and intervention settings.

References

[1] Erfan Al-Hossami and Razvan Bunescu. 2026. McMining: Automated Discovery of Misconceptions in Student Code. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*. 160–178. doi:10.18653/v1/2026.eacl-short.10

[2] Amjad Altadmri and Neil C. C. Brown. 2015. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 522–527. doi:10.1145/2676723.2677258

[3] Morten Bastian and Andreas Mühling. 2025. Misconceptions in Programming: Intuitive Reasoning and Tracing Task Performance Across Experience Levels. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V. 1*. 141–154.

[4] Neil C. C. Brown and Amjad Altadmri. 2014. Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*. 43–50. doi:10.1145/2632320.2632343

[5] Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L Santos, and Matthias Hauswirth. 2021. A curated inventory of programming language misconceptions. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 380–386.

[6] Albert T. Corbett and John R. Anderson. 1995. Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modeling and User-Adapted Interaction* 4, 4 (1995), 253–278. doi:10.1007/BF01099821

[7] Stephen H. Edwards and Krishnan P. Murali. 2017. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM*

Conference on Innovation and Technology in Computer Science Education. 188–193. doi:10.1145/3059009.3059055

[8] Aritra Ghosh, Jay Raspat, and Andrew S. Lan. 2021. Option Tracing: Beyond Correctness Analysis in Knowledge Tracing. In *Artificial Intelligence in Education*. 137–149. doi:10.1007/978-3-030-78292-4_12

[9] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. 41–63. doi:10.1145/2858796.2858798

[10] Naiming Liu, Zichao Wang, Richard Baraniuk, and Andrew Lan. 2022. Open-ended Knowledge Tracing for Computer Science Education. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 3849–3862. doi:10.18653/v1/2022.emnlp-main.254

[11] Jose Llanos, Víctor A Bucheli, and Felipe Restrepo-Calle. 2023. Early prediction of student performance in CS1 programming courses. *PeerJ Computer Science* 9 (2023), e1655.

[12] Liting Lyu, Zhifeng Wang, Haihong Yun, Zexue Yang, and Ya Li. 2022. Deep knowledge tracing based on spatial and temporal representation learning for learning performance prediction. *Applied Sciences* 12, 14 (2022), 7188.

[13] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J. Guibas, and Jascha Sohl-Dickstein. 2015. Deep Knowledge Tracing. In *Advances in Neural Information Processing Systems 28*. 505–513.

[14] Thomas W. Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M. Kazerouni, Brett A. Becker, Andrew Petersen, Luke Gusukuma, Stephen H. Edwards, and David Babcock. 2020. ProgSnap2: A Flexible Format for Programming Process Data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 118–124. doi:10.1145/3341525.3387373

[15] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–24.

[16] Sue Sentance, Jane Waite, and Maria Kallia. 2019. Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education* 29, 2-3 (2019), 136–176.

[17] Shuanghong Shen, Qi Liu, Zhenya Huang, Yonghe Zheng, Minghao Yin, Minjuan Wang, and Enhong Chen. 2024. A survey of knowledge tracing: Models, variants, and applications. *IEEE Transactions on Learning Technologies* 17 (2024), 1858–1879.

[18] Yang Shi, Krupal Shah, Wengran Wang, Samiha Marwan, Poorvaja Penmetsa, and Thomas W. Price. 2021. Toward Semi-Automatic Misconception Discovery Using Code Embeddings. In *Proceedings of the 11th International Learning Analytics and Knowledge Conference*. 606–612. doi:10.1145/3448139.3448205

[19] Haoye Tian, Kui Liu, Abdoul Kader Kaboré, Anil Koyuncu, Li Li, Jacques Klein, and Tegawendé F Bissyandé. 2020. Evaluating representation learning of code changes for predicting patch correctness in program repair. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. 981–992.

[20] Ashok Kumar Veerasamy, Daryl D'Souza, and Mikko-Jussi Laakso. 2016. Identifying Novice Student Programming Misconceptions and Errors From Summative Assessments. *Journal of Educational Technology Systems* 45, 1 (2016), 50–73. doi:10.1177/0047239515627263