Do LLM Student Models Track the Student or the Situation? A Matched-Peer Study in Novice Programming

Shlok Shah
shlok10@student.ubc.ca
University of British Columbia, Okanagan
Kelowna, Canada

Abdallah Mohamed
abdallah.mohamed@ubc.ca
University of British Columbia, Okanagan
Kelowna, Canada

Abstract

Educators increasingly use large language models (LLMs) as student models: to summarize a learner's work, describe their understanding, and predict their next moves. For educators evaluating these tools, a fragile assumption sits underneath: a model's response can sound tailored to a student while only reflecting the programming situation. We ask whether an LLM's descriptions and predictions about novice Python programmers are tied to the genuinely individual student.

We study introductory Python submissions from CodeBench, using 117 narrative cases and a matched-peer prediction audit of 310 paired comparisons. Written summaries appear personalized: varied, behaviorally grounded, with little template language. A stricter prediction test reveals a gap. For students matched on the same exercise assessment state, predictions without process traces are roughly twice as similar in predicted activity shape (local run-count agreement) as students' real next submissions, while other granularities show smaller or uncertain effects. Adding traces reduces this collapse and adds coarse repair-magnitude and structural signal, but reliable trace-added gains for code details, edit locations, or identifiers remain absent.

For educators evaluating LLM-powered tools, personalized-sounding summaries are insufficient evidence of student-specific modeling. We propose identity perturbation under matched assessment state: pair students at the same exercise and test-outcome state, then test whether predictions are correctly targeted, not merely varied. Personalization claims should specify what student-specific signal is captured, from coarse behavior to fine-grained code, so educators and tool designers can judge when LLM-based personalization is warranted.

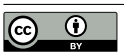
CCS Concepts

• **Social and professional topics** → **Computer science education**; • **Information systems** → *Learning management systems*; • **Computing methodologies** → *Natural language processing*.

Keywords

LLMs, student modeling, novice programming, personalization

ACM Reference Format:
Shlok Shah and Abdallah Mohamed. 2026. Do LLM Student Models Track the Student or the Situation? A Matched-Peer Study in Novice Programming. In *Proceedings of the 2nd ACM Virtual Global Computing Education Conference V.1 (SIGCSE Virtual 2026), November 12–15, 2026, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3795867.3830982



SIGCSE Virtual 2026, Virtual Event, USA.

ACM ISBN 979-8-4007-2506-7/2026/11
https://doi.org/10.1145/3795867.3830982

1 Introduction

Large language models are increasingly used as informal student models in computing education: they summarize a learner's current work, infer likely misunderstandings, recommend next steps, and predict what a student may do next. For instructors and tool designers, this trend raises an immediate evaluation problem: how to tell when apparently personalized output is actually tied to the individual learner. These uses are attractive because they appear to convert messy traces of programming activity into language that an educator can inspect. They are also risky. A summary can sound highly individualized while mostly reflecting the exercise, the current failing tests, or common novice repairs. If such state-level guidance is presented as learner-specific, instructors may over-trust it when triaging work or planning interventions. We test whether personalization claims are warranted; we do not test whether personalization improves learning outcomes.

This paper studies whether an LLM response tracks the particular novice programmer or only the programming situation. We use *programming situation* to mean the exercise and the current assessment evidence available at the focal attempt, especially the test-level pass/fail state. We use *student-specific targeting* more narrowly: a prediction for student A should fit A's observed next submission better than the observed next submission of a matched peer B who is in the same assessed state. This definition is intentionally output-facing. It does not require us to infer an internal cognitive representation, and it avoids a common validation shortcut: treating plausible, varied prose as evidence of personalization.

A matched assessment state is not a claim that students have identical source code or identical histories. It fixes the class, task, exercise, and row-local test pass/fail vector while allowing current code and process history to vary as possible differentiating evidence. Here, *identity* means focal-student versus matched-peer identity in the dataset; we do not perturb or evaluate demographic identity attributes.

We therefore separate two senses of personalization. The first is *surface narrative variation*: whether written summaries look varied, behaviorally grounded, and non-templated. The second is *prediction-level targeting*: whether predictions about a student's next programming move are more accurate for that student than for a matched peer in the same assessed state. The latter is stricter. A model that predicts the same next move for every matched peer may still write individualized prose, but it has not demonstrated

the kind of student-specific model educators need when evaluating personalization claims.

Our method is a matched-peer identity perturbation audit. For each retained assessment state, we compare the prediction generated for a focal student with both the focal student's observed next move and a matched peer's observed next move. If the output mainly reflects the shared programming situation, it should fit the matched peer about as well as the focal student or become more homogeneous than the peers' real next moves. If process traces carry student-specific signal, Full trace predictions should improve under self-vs-peer scoring and should reduce this peer-reality over-collapse. The protocol deliberately separates two claims that are often conflated: producing different outputs for different histories, and producing outputs that are correctly targeted to the right learner.

We make three contributions:

- We introduce identity perturbation under matched assessment state as a validation protocol for LLM student-modeling claims in novice programming, operationalizing personalization as correct targeting rather than output variation alone.
- We analyze 117 narrative cases and 310 directed matched-peer comparisons of CodeBench Python submissions.
- We show a gap between personalized-sounding summaries and prediction-level targeting: traces reduce the clearest generic-collapse pattern and improve targeting of coarse repair-shape signal, but trace-added content, location, and identifier gains remain unsupported.

2 Related Work

Student modeling has long framed learning systems around latent or observed representations of a learner's current knowledge, strategy, or likely future performance. Classical knowledge tracing models estimate mastery from sequences of observed responses [3], and later neural approaches use interaction histories to predict future performance [11]. Programming-specific learning analytics has used code snapshots, online judge logs, submissions, test outcomes, and keystroke-level process data to characterize novice behavior [1, 10]. The CodeBench environment is especially relevant because it captures both execution/submission logs and IDE interaction data for introductory programming and has released a public learning-analytics dataset [2]. Our test complements these learner-state and future-performance objectives with an assignment check: within a shared assessed state, is the prediction better matched to the learner whose history generated it?

Recent LLM-based educational systems extend this tradition by using language models to synthesize student behavior, generate feedback, or predict future work. Automated feedback for programming has a long history of using student code and errors as evidence [6]; LLMs have now been studied for generating programming exercises and code explanations [16], supporting novice learners with code generators [5], and providing guarded help in programming classes [8]. Data-driven programming-hint systems are a close precursor because they generate next-step guidance from a learner's current code state and historical solution paths [12, 13]. Recent LLM next-step hint work similarly evaluates whether generated hints are specific to a student's code and approach while documenting risks such as misleading or insufficient hints [14]. More directly, LLM-SS studies in-context student modeling by synthesizing student attempts in visual programming [9], and TIKTOC models open-ended coding tasks using test case information and code prediction objectives [4]. Adjacent writing-feedback work reports that adding keylogs and time-stamped revision snapshots can make LLM-generated feedback more process-sensitive [17]. These studies motivate using LLMs as student models, but they also raise a validation problem: improvements in plausible explanation, guarded support, or aggregate prediction do not necessarily show that a model distinguishes one matched student from another in the same assessed programming situation.

This prior work motivates our framing but leaves a specific validation gap. Studies can demonstrate useful feedback, plausible generated behavior, or aggregate prediction without testing whether two students in the same assessed programming situation receive predictions targeted to the correct individual. Outside education, personalization benchmarks evaluate whether LLM outputs benefit from user-specific histories or profiles [15]. Counterfactual evaluation traditions similarly ask whether outputs remain invariant, or change appropriately, when identity-relevant variables are counterfactually changed while other state variables are held fixed [7]. Our work adapts this logic to novice programming. We do not infer the model's internal cognitive representation; we ask whether externally visible predictions remain correctly targeted when matched peers share the same exercise assessment state.

3 Research Questions

The research questions deliberately move from weak to strong evidence: first surface-level narrative variation, then generic collapse under matched state, then the granularity of any trace-supported targeting. We ask three questions.

RQ1: Do LLM-written summaries of novice programming work avoid literal templating and surface-level repetition across students?

RQ2: At what granularity do predictions for students matched on the same current assessment state collapse toward a generic next move for the state?

RQ3: At what granularity does process trace information improve prediction targeting: coarse repair shape, fine-grained code content, edit location, or identifier choice?

4 Method

4.1 Data Setting

We use CodeBench, an online judge and embedded IDE used in introductory programming courses at the Federal University of Amazonas [2, 10]. CodeBench records source submissions, execution outcomes, test-level pass/fail feedback, and IDE activity. Our analyses use Python submissions from introductory programming exercises. We report only anonymized identifiers and aggregate behavior.

The study has two separately constructed parts with different evidentiary roles. The narrative analysis uses 117 cases selected across available transition depths; it asks whether short summaries avoid literal templating and surface-level repetition. These cases

are not the denominator for the prediction audit. The stricter prediction audit fixes one decision point: a *prediction row* contains a focal student's third visible submission and the observed fourth submission as its target. This choice balances history against cohort yield. Three submissions give the model multi-attempt evidence, while requiring a fourth target already reduced 73,635 candidates with both logs to 4,445 parseable, depth-eligible candidates before matching. A *matched assessment-state scope* groups rows from the same class, task, and exercise with the same current row-local test-level pass/fail vector. Across four CodeBench semesters (2022-2, 2023-1, 2023-2, and 2024-1), the final set contains 91 scopes, 209 rows per condition, and 310 directed focal-peer comparisons. Figure 1 summarizes the selection and scoring flow. In both parts, the substantive question is whether output is tied to the student rather than only to the situation.

4.2 Ethics and Data Use

We use de-identified CodeBench records and report only aggregate behavior. LLM requests contained the exercise context, submitted code, execution feedback, and available trace evidence needed for the prediction task; they did not include student names, demographic attributes, or institution-specific identifiers beyond anonymized dataset keys. External LLM processing used only these de-identified records and aggregate outputs.

4.3 Narrative Audit Procedure

For 117 matched transitions, the narrative analysis uses the top-ranked `student_state_summary` from paired Full trace and No trace runs. The prompt asked for a cautious instructor-facing account of the student's current likely state and likely next repair move, grounded only in evidence available up to attempt n. It explicitly prohibited using information from attempt $n+1$ or later and required uncertainty to be expressed through multiple hypotheses.

We audit the narratives in two ways. First, to detect template-like prose, we group Full trace summaries by exercise, identify repeated 3–6 word phrases appearing in at least half of summaries for that exercise, prune overlapping phrases, and compute the fraction of summary tokens covered by those repeated phrases. Exercises with fewer than three distinct students are excluded from this template-fraction calculation. Second, to compare Full trace and No trace summaries for the same transition, we compute corpus-level TF-IDF vectors from all matched summaries and report cosine similarity between paired summaries. This audit is intentionally textual: it tests surface variation and template reuse, not whether the predicted next code is correctly targeted.

4.4 Prediction Conditions

For each prediction row, the model saw the exercise context and submitted-code/execution evidence through the focal attempt; *IDE interaction traces* varied by condition. By IDE interaction traces, we mean the pre-focal IDE edits, local-run behavior, code snapshots, and submit feedback that record how the student worked before the prediction point. For reporting, we call the condition without these traces *No trace* and the condition with them *Full trace*.

No trace includes submitted-code and execution/test information for the visible attempts, but omits IDE interaction traces. Thus, *No trace* means no interaction-trace evidence, not no prior submission history.

Full trace includes the same submitted-code and execution/test information plus only process evidence available before the focal prediction point. For each visible attempt, the prompt includes attempt-start code; raw CodeMirror edit segments rendered as tabular change records with timestamps, line/character coordinates, inserted and removed text, and edit origin; local-run raw output and code snapshots; and submit feedback with code snapshots. It excludes the observed next submission and all events after the focal attempt. This condition tests whether the additional process trace supports student-specific targeting.

Both conditions used `gpt-5.4`, resolved by the API to snapshot `gpt-5.4-2026-03-05`, with `reasoning.effort=xhigh`. This setting allocates greater internal reasoning effort and was chosen to give the model a fair opportunity to use long, messy trace evidence. The two conditions use the same prediction task and scoring pipeline; the ablation is whether process-trace evidence is present. Requests were submitted through the Responses batch endpoint in April 2026 using API-default sampling parameters; all 209 requests per condition completed without API-reported failures. A strict JSON schema requires exactly three hypotheses whose probabilities sum to 1.0. Each includes an estimated probability, predicted next code, and a predicted trajectory of one to eight coarse steps (`edit`, `local_run`, or `submit`) with line ranges. We evaluate the top-ranked prediction, defined by the model's highest estimated probability, against the observed next submission.

4.5 Matched-Peer Identity Perturbation

The core design holds the assessed situation fixed and changes the student identity at evaluation time. Students are eligible matched peers when they share the same class, task, exercise, and current row-local test-level pass/fail vector at the focal attempt. We call this a *matched assessment state*. The match intentionally does not require identical source code or identical histories: those differences are available differentiating evidence, and normalized code-distance filtering is treated as sensitivity analysis. A group with m students contributes $m(m-1)$ directed focal-peer comparisons and is retained only if $m \geq 2$.

For a directed pair $A \rightarrow B$, the prediction generated from A's context is scored against both A's and B's observed next submissions. Direction matters because B has a separate prompt context, prediction, and reverse comparison.

One retained pair illustrates the design. At their third submissions, A and B were on the same exercise and failed the same two tests, but their code and histories differed: A hard-coded the current year, whereas B read it as input. In their observed fourth submissions, A corrected the foundation-year constant in the subtraction, while B removed the input prompt and revised output wording. We score the prediction generated from A's context against both observed fourth submissions; the reverse direction uses B's prediction. This example explains the scoring operation, not the aggregate evidence.

Self-vs-peer targeting therefore asks whether a prediction maps to the right student, not merely whether it is plausible for the shared state. For metric s and condition c, it is $s(\hat{y}_A^c, y_A) - s(\hat{y}_A^c, y_B)$.

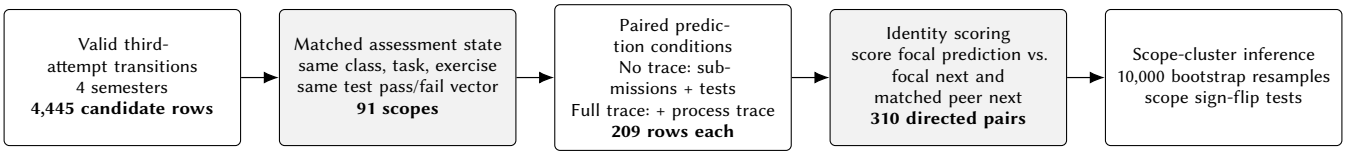


Figure 1: Prediction-audit flow for identity perturbation under matched assessment state. CodeBench transitions are filtered to valid third-attempt rows, grouped by class, task, exercise, and row-local test pass/fail vector, and retained when a scope contains at least two students. The retained audit set has 91 matched scopes, 209 prediction rows per condition, and 310 directed focal-peer comparisons. Each focal prediction is scored against both the focal student's observed next submission and a matched peer's observed next submission; inference is clustered by matched assessment-state scope.

Table 3 reports its within-condition means and the Full-minus-No trace difference.

Peer-reality over-collapse instead asks whether matched peers' predictions are more similar than their observed next submissions. It compares $s(\hat{y}_A^c, \hat{y}_B^c)$ with $s(y_A, y_B)$; large positive values indicate compression toward a generic "student in this situation." Table 2 reports both similarities and the Full-minus-No trace change.

4.6 Metrics and Inference

We use metrics that capture different granularities of next-move targeting. Fine-grained metrics test exact code, aligned content, identifier choices, and edit locations. Coarser metrics test repair magnitude, structural change, activity shape, and action-sequence similarity. This distinction matters because a trace condition can reduce generic behavior at a coarse level without demonstrating fine-grained student-specific code targeting. Bounded code gain is $(d_{\text{copy}} - d_{\text{pred}})/\max(d_{\text{copy}}, d_{\text{pred}})$ using normalized edit distance to the observed next code, so 1 is exact, 0 is no better than copying the current code, and negative is worse than copying. For example, a bounded code gain of 0.5 means the prediction closes half the available distance from copying the current code to the observed next code. Structural lift subtracts the current-code-to-target token similarity from the predicted-code-to-target token similarity. Aligned content F1 aligns predicted and observed repair hunks by line-window overlap, then averages token F1 over aligned hunks while scoring unaligned hunks as zero. Local run-count agreement is $1 - |p - o|/\max(p, o, 1)$ for predicted and observed pre-submit local-run counts, a graded activity-shape metric rather than semantic code targeting. Table 1 summarizes the metric interpretation.

Pairwise comparisons are not independent because many directed pairs share the same exercise state. Most retained scopes contain two matched students (71 of 91 scopes); 16 scopes contain three students, one contains four students, and three contain five students. We report pair-weighted directed-comparison means and aggregate uncertainty at the matched-state scope level using a scope-cluster bootstrap with 10,000 samples. We also report two-sided scope sign-flip randomization p-values. We treat p-values as descriptive evidence rather than as proof of a latent cognitive mechanism.

Table 1: Metric granularity for RQ2 and RQ3. Whole-code metrics compare the entire submitted file; content metrics compare tokens within aligned repair hunks. Coarse metrics can improve without establishing fine-grained targeting.

Metric family	Metric	Targeting level
Exact code	Exact next-code match	whole submitted file
Content	Aligned content F1	aligned repair-hunk tokens
Identifiers	Identifier content F1	naming choices
Location	Windowed footprint F1	edit footprint
Location	Edit-region overlap	edit region
Repair shape	Bounded code gain	repair magnitude
Repair shape	Structural lift over copy	structural change
Trajectory	Local run agreement	activity shape
Trajectory	Trajectory alignment	action sequence

5 Results

5.1 Narratives Avoid Literal Templates

The 117 narrative cases show why prose-only validation is tempting but weak. Summaries are behaviorally specific and rarely templated. A slot-filling audit across 14 exercises found that only 2.3% of summary text belonged to repeated template phrases, with the remaining 97.7% varying by case. Comparing Full trace and No trace summaries for the same transition, 115 of 117 pairs had TF-IDF cosine below 0.5, with mean cosine 0.17. These textual indicators help explain why such summaries may plausibly read as individualized in ordinary review.

This answers RQ1 at the surface-text level: the summaries are not simply literal templates or repeated boilerplate. It does not show that the predicted next move is targeted to the right student: two matched students can receive different, grounded narratives even when a prediction would fit either student's next move equally well. The matched-peer audit therefore tests the stronger claim needed for student-specific modeling.

5.2 No-Trace Predictions Over-Collapse on Activity Shape

The tables test different failures. Peer-reality over-collapse asks whether matched peers' predictions are more alike than their real next submissions; self-vs-peer targeting asks whether a prediction fits its focal student better than a peer. Traces can improve the first without improving the second. Peer-reality over-collapse shows

Table 2: Peer-reality over-collapse under matched assessment state. Full and No trace are prediction-to-prediction similarities among matched peers; Real is similarity between the peers' observed next submissions. The reported change is Δ_{over} = (Full−Real)−(No trace−Real) = Full−No trace; negative values mean traces reduce over-collapse.

Metric	Full	No	Real	Δ_{over}	95% CI
Local-run agree.	.582	.916	.404	−.334	[−.444, −.217]
Edit region	.401	.429	.433	−.028	[−.064, .006]
Trajectory	.397	.394	.314	.003	[−.069, .080]
Exact code	.000	.006	.000	−.006	[−.022, .000]

the clearest failure mode. Without process traces, predictions for matched peers were much more similar than the students' real next submissions. The strongest example is local run-count agreement: No trace predictions agreed with each other at 0.916, while matched students' real next submissions agreed at 0.404. That is more than twice the real peer similarity. Full trace predictions were still more similar than reality, but less extremely so: 0.582 versus 0.404. The difference in over-collapse, $(.582 - .404) - (.916 - .404)$, was −0.334 (95% CI [−0.444, −0.217], $p < .001$).

This result supports RQ2, but only at the granularity where the effect is observed. The No trace model often behaves as if the assessed exercise state is enough to determine the next activity shape. Traces reduce that homogenization, especially for local-run behavior. Other over-collapse rows are small or uncertain, so we do not claim broad code-level de-collapse from this table alone.

Table 3: Self-vs-peer targeting under matched assessment state. Full and No trace are within-condition mean self-minus-peer discrimination; positive values mean the prediction fits the focal next move better than a matched peer's next move. Δ is Full trace minus No trace; positive Δ means traces improve discrimination.

Metric	Full	No	Δ [95% CI]	p
Code gain	.614	.494	.120 [.056, .187]	< .001
Struct. lift	.295	.277	.018 [.008, .029]	.0015
Edit region	.194	.163	.031 [−.003, .066]	.093
Trajectory	.121	.099	.022 [−.009, .056]	.193
Exact code	.077	.055	.023 [−.020, .065]	.404
Content F1	.112	.103	.008 [−.023, .040]	.608
Identifier F1	.127	.122	.005 [−.027, .037]	.768
Footprint F1	.102	.108	−.006 [−.035, .021]	.685
Run agree.	.055	.011	.044 [−.036, .133]	.295

5.3 Trace Adds Coarse Repair-Shape Signal, Not Fine-Grained Signal

Self-vs-peer targeting is a discrimination test, not an aggregate next-code accuracy test. A positive within-condition value means a prediction fits the focal student's next move better than a matched peer's next move; a positive Full-minus-No trace value means traces improve that discrimination. Under this test, Full trace shows two positive cross-condition effects, both coarse. It improves structural lift over copy by 0.018 (95% CI $[0.008, 0.029]$, $p = .0015$) and bounded code gain over copy by 0.120 (95% CI $[0.056, 0.187]$, $p < .001$). These effects indicate relative improvement over No trace in repair-shape or repair-magnitude discrimination; they do not by themselves establish fine-grained semantic targeting.

The nulls are as important as the positives, but they are nulls for trace-added discrimination rather than proof of no absolute self-vs-peer signal. Because both conditions include the focal student's submitted code and test feedback, positive within-condition means may reflect targeting already available from the submitted-code state. What Table 3 shows is that adding process traces does not reliably improve aligned content F1, identifier content F1, windowed footprint F1, exact next-code match, trajectory alignment, local-run agreement, or edit-region overlap beyond that baseline. The two positive effects survive a conservative Bonferroni threshold of .05/9 across the nine reported metrics. Thus, for RQ3, trace evidence improves some coarse repair-shape discrimination, but we do not find reliable trace-added gains for fine-grained code content, location, or identifier choices that distinguish matched students.

Because an assessment-state match does not require identical current code, we filtered pairs post hoc by normalized Levenshtein distance between focal codes. Thresholds 0.02, 0.25, and 0.50 retained, respectively, 2, 132, and 284 directed pairs from 1, 37, and 86 scopes. The 0.02 layer was too thin for scope-cluster inference. Across estimable thresholds, code-gain effects were positive with sign-flip evidence below .05 from 0.25 onward; structural-lift confidence intervals were positive from 0.25 onward, and local-run over-collapse remained negative. We treat this as robustness analysis, not a second estimand or a claim of identical code.

6 Discussion

6.1 Personalized Prose Is Not Validation

Varied, grounded narratives can coexist with weak student-specific targeting. For educators, a plausible summary may still help summarize situations or triage common states, but it should not be treated as evidence of learner-specific next-move signal. Personalization claims should name the supported level: activity, repair magnitude, structure, edit location, identifier choice, or code content.

6.2 What Traces Buy

Trace access made predictions less homogeneous and improved coarse repair-shape targeting, not reconstruction of the student's exact next code. Distinguishing a small local patch from a structural repair may still support instructor reflection or triage, although we did not evaluate instructor use. The evidence therefore supports "uses student trace data" more strongly than "models the student."

6.3 Identity Perturbation as a Validation Step

Matched-peer identity perturbation is deliberately simple. It asks whether predictions remain targeted when the situation is held approximately fixed. This makes it attractive as a validation step for LLM-powered educational tools. If a system claims to personalize, it should be evaluated not only on aggregate accuracy or human-rated plausibility, but also on whether its output changes in the right way

across matched students. The key methodological point is to test de-homogenization and correct targeting separately: a model can become less generic without becoming reliably student-specific.

In practice, an evaluation should identify matched-state scopes in deployment data, score each prediction against both the focal student's next work and a matched peer's next work, and report the granularity at which targeting holds. A tool that passes for repair magnitude but not for identifier choice or edit location may still be useful, but its personalization claim should be scoped to repair shape rather than to modeling the student's exact next code.

The design also separates generic outputs shared across a state from varied but wrong-targeted outputs that do not map to the right learner. Both undermine personalization, but they require different redesigns.

7 Limitations and Validity Threats

Construct validity. Our matched-state design controls the current exercise assessment state, not the full programming state. Students in a match group share the same class, task, exercise, and row-local pass/fail vector, but they may differ in source code, trace history, and prior learning experience. This is why we describe the primary denominator as an assessment-state match and treat normalized code-distance filtering as sensitivity analysis rather than as the paper's main estimand. The narrative audit is also a limited construct: low template use and low TF-IDF similarity show prose variation, not true personalization. The prediction metrics are behavioral output measures. They can show whether predictions fit the observed next move, but they cannot reveal the LLM's internal representation of the learner. We therefore do not claim that the LLM lacks an internal student model; we claim that these outputs do not show reliable trace-added fine-grained student-specific targeting under matched assessment state.

Internal validity. The Full trace condition exposes prior editing and local-run behavior, so improvements in local-run agreement are partly expected and should be interpreted as activity-shape de-homogenization rather than as evidence of semantic learner modeling. The No trace condition still includes submitted-code history and test feedback for visible attempts, so it is not an information-free baseline. The prediction audit uses a fixed third-attempt setup and a top-ranked prediction view; other deployment choices, such as using lower-ranked hypotheses, different prompts, or different output formats, may shift absolute scores. Because requests used API-default sampling and a single batch run, the reported uncertainty quantifies variation over matched assessment-state scopes, not run-to-run LLM stochasticity or future model-version changes. The code-distance sweep helps check whether results depend on near-identical current code, but the thinnest threshold is too sparse for scope-cluster inference.

External validity. Because the empirical evidence comes from one LLM family, one programming environment, and introductory Python exercises from CodeBench, we frame the results primarily as a demonstration of the matched-peer identity perturbation protocol and as evidence for this setting, not as a general claim about all LLM student models. The final prediction audit emphasizes common assessed states where at least two students can be matched, so it does not cover every kind of novice programming behavior. The narrative and prediction analyses are two analytic lenses on the same study family: the narrative audit establishes personalized-looking prose, while the matched-peer prediction audit tests a stricter targeting claim. We do not generalize from this evidence to all LLM tutors, all programming courses, or all forms of educational personalization.

8 Conclusion

LLM student models can sound personalized without demonstrating student-specific prediction. In CodeBench, No trace predictions over-collapse most clearly in activity shape. Traces reduce that collapse and improve some coarse repair-shape discrimination, but do not reliably add targeting of code content, edit locations, or identifiers.

This audit is best used as a validation layer, not as a replacement for studies of usefulness or learning outcomes. A system may still be useful while failing fine-grained targeting, but it should then be described as state-aware rather than strongly student-specific. Future evaluations should vary history depth, model snapshots, and matching strictness while reporting how many students and matched scopes remain at each layer. Doing so makes the history-versus-yield tradeoff explicit and separates evidence for coarse state adaptation from evidence of learner-specific modeling.

For computing education researchers and tool designers, the implication is direct: validate personalization under perturbation. Pair students at the same current assessment state, compare predictions against matched peers, and specify the granularity of student-specific signal that is actually supported. This separates outputs that merely vary across histories from outputs that are warranted as learner-specific.

References

[1] Paulo Blikstein. 2011. Using Learning Analytics to Assess Students' Behavior in Open-Ended Programming Tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*. Association for Computing Machinery, New York, NY, USA, 110–116. doi:10.1145/2090116.2090132

[2] Flávio J. M. Coelho, Elaine H. T. Oliveira, Filipe D. Pereira, David B. F. Oliveira, Leandro S. G. Carvalho, Eduardo J. P. Souto, Marcela Pessoa, Rafaela Melo, Marcos A. P. de Lima, and Fabiola G. Nakamura. 2023. Learning Analytics em Cursos de Introdução à Programação: Uma Mostra da Universidade Federal do Amazonas. *Revista Brasileira de Informática na Educação* 31 (2023), 1089–1127. doi:10.5753/rbie.2023.3334

[3] Albert T. Corbett and John R. Anderson. 1994. Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modeling and User-Adapted Interaction* 4, 4 (1994), 253–278. doi:10.1007/BF01099821

[4] Zhangqi Duan, Nigel Fernandez, Alexander Hicks, and Andrew S. Lan. 2025. Test Case-Informed Knowledge Tracing for Open-ended Coding Tasks. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference* (Dublin, Ireland). Association for Computing Machinery, New York, NY, USA, 238–248. doi:10.1145/3706468.3706500

[5] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. doi:10.1145/3544548.3580919

[6] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* 19, 1, Article 3 (2018), 43 pages. doi:10.1145/3231711

[7] Matt J. Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual Fairness. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., Red Hook, NY, USA, 4066–4076. https://proceedings.neurips.cc/paper_files/paper/2017/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html

[8] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2024. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Association for Computing Machinery, New York, NY, USA, Article 8, 11 pages. doi:10.1145/3631802.3631830

[9] Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. 2024. Large Language Models for In-Context Student Modeling: Synthesizing Student's Behavior in Visual Programming. In *Proceedings of the 17th International Conference on Educational Data Mining* (Atlanta, Georgia, USA). International Educational Data Mining Society, Atlanta, Georgia, USA, 341–348. doi:10.5281/zenodo.12729830

[10] Filipe Dwan Pereira, Elaine H. T. Oliveira, David B. F. Oliveira, Alexandra I. Cristea, Leandro S. G. Carvalho, Samuel C. Fonseca, Armando Toda, and Seiji Isotani. 2020. Using Learning Analytics in the Amazonas: Understanding Students' Behaviour in Introductory Programming. *British Journal of Educational Technology* 51, 4 (2020), 955–972. doi:10.1111/bjet.12953

[11] 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*, Vol. 28. Curran Associates, Inc., Red Hook, NY, USA, 505–513. https://papers.nips.cc/paper/5654-deep-knowledge-tracing

[12] Thomas W. Price, Yihuan Dong, and Tiffany Barnes. 2016. Generating Data-Driven Hints for Open-Ended Programming. In *Proceedings of the 9th International Conference on Educational Data Mining*. International Educational Data Mining Society, Raleigh, NC, USA, 191–198. https://www.educationaldatamining.org/EDM2016/proceedings/paper_33.pdf

[13] Kelly Rivers and Kenneth R. Koedinger. 2017. Data-Driven Hint Generation in Vast Solution Spaces: A Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 37–64. doi:10.1007/s40593-015-0070-z

[14] Lianne Roest, Hieke Keuning, and Johan Jeuring. 2024. Next-Step Hint Generation for Introductory Programming Using Large Language Models. In *Proceedings of the 26th Australasian Computing Education Conference*. Association for Computing Machinery, New York, NY, USA, 144–153. doi:10.1145/3636243.3636259

[15] Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. 2024. LaMP: When Large Language Models Meet Personalization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 7370–7392. doi:10.18653/v1/2024.acl-long.399

[16] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research*. Association for Computing Machinery, New York, NY, USA, 27–43. doi:10.1145/3501385.3543957

[17] Samra Zafar, Shifa Yousaf, and Muhammad Shaheer Minhas. 2025. "Can You See Me Think?" Grounding LLM Feedback in Keystrokes and Revision Patterns. arXiv:2508.13543 [cs.HC] doi:10.48550/arXiv.2508.13543