

PARASTUDENT: Closing the Sim2Real Gap in User Simulators for AI Tutor Evaluation

Rose Niousha Mihran Miroyan Abigail O’Neill Joseph E. Gonzalez
Gireeja Ranade John DeNero Narges Norouzi

University of California, Berkeley
{rose.n, miroyan.mihran, abbyoneill, jegonzal,
ranade, denero, norouzi}@berkeley.edu

Abstract

Evaluating Artificial Intelligence (AI) tutor feedback before deployment requires anticipating student engagement, typically assessed through real interaction data. We introduce PARASTUDENT, a fine-tuning framework for simulating novice programming revisions to support AI tutor evaluation. Compared with prompted baselines, PARASTUDENT’s revisions more closely match real student code distributions across functional, stylistic, and semantic metrics. Our best variant achieves AUCs of 0.80 for both feedback relevance and successful uptake when distinguishing streams with real engagement above versus at or below the median, while prompted baselines remain near chance on successful uptake. These findings demonstrate the promise of simulated engagement for pre-deployment feedback triage.

1 Introduction

As Large Language Models (LLMs) are increasingly deployed as interactive assistants, evaluation is shifting beyond isolated turn-level response quality toward measuring how users engage with assistants over time (Zheng et al., 2023; Liao and Xiao, 2023; Mysore et al., 2025; Li et al., 2025). However, real user interaction data is typically only available after deployment, which can make evaluation slow, expensive, and potentially risky in high-stakes domains.

To address this limitation, recent work has explored LLM-based user simulators for pre-deployment evaluation (Yao et al., 2025; Zhang et al., 2025a; Qian et al., 2025). While such an approach is promising, for simulation-based evaluation to be truly useful, the simulator must reproduce realistic user behavior and support the same evaluation decisions as real interaction data (Suh et al., 2026; Zhou et al., 2026).

Artificial Intelligence (AI) tutoring is a particularly suitable setting for studying simulation-based

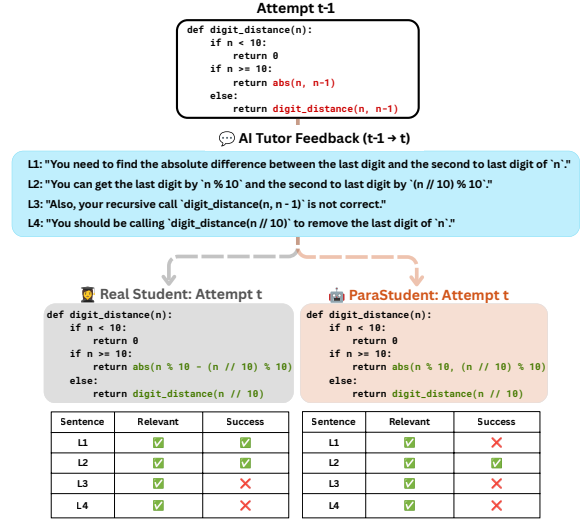


Figure 1: **Overview of real student (left) and PARASTUDENT (right) edits conditioned on identical prior code and AI tutor feedback.** Each feedback sentence is labeled by whether it influenced the next attempt and whether it was applied correctly; red marks erroneous expressions in attempt $t-1$, green the revised ones in attempt t . Both apply every sentence only partially, and reproducing this incomplete uptake is what enables engagement-based evaluation before deployment.

evaluation (Yuan et al., 2026). Because effective tutoring depends not only on what the tutor says, but also on how students engage with that feedback (Niousha et al., 2026), existing AI tutor benchmarks that evaluate tutor responses primarily through static pedagogical qualities such as correctness, style, or solution leakage (Macina et al., 2025; Srinivasa et al., 2025; Maurya et al., 2025) provide only a partial view of tutor effectiveness. Furthermore, pre-deployment evaluation is especially important in education, where deploying ineffective AI tutors not only results in poor user experience but can also directly harm student learning (Jin et al., 2025; Ezzaki et al., 2025).

In particular, AI programming tutors, which are increasingly used by novice learners (Liu

et al., 2024; Kazemitabaar et al., 2024; Zamfirescu-Pereira et al., 2025), provide a strong setting for simulation-based AI tutor evaluation because learning unfolds through iterative code revisions that provide observable traces of how tutor feedback shapes subsequent student behavior. In addition, prior work shows that simulating programming behavior is more feasible than simulating open-ended dialogue with LLM-based user simulators (Mehri et al., 2026).

We introduce PARASTUDENT (Figure 1), a framework for simulating novice programming revisions to evaluate AI tutor feedback before deployment. PARASTUDENT fine-tunes LLMs conditioned on students’ code submissions and AI tutor feedback, with the goal of capturing not only plausible novice code but also how specific feedback shapes subsequent student behavior. We first validate the simulator against real student behavior across functional, stylistic, and semantic dimensions, motivated by prior educational literature, comparing it with existing approaches for simulating novice programmers. Having established this fidelity, we then demonstrate a downstream use case: engagement-based evaluation of AI tutors. Overall, PARASTUDENT more closely matches real student code than prompted baselines, both distributionally and at the level of paired submissions. Our best variant achieves AUCs of approximately 0.80 for both feedback relevance and successful uptake when distinguishing interaction streams above versus at or below the median real student engagement. These findings demonstrate the promise of simulated student engagement for pre-deployment triage of AI tutor feedback.¹

Our contributions are threefold:

1. We introduce a fine-tuning method for simulating novice programming revisions conditioned on prior submissions and AI tutor feedback.
2. We propose a multi-dimensional evaluation framework grounded in programming education for validating simulator fidelity across functional, stylistic, and semantic dimensions.
3. We show that distributional fidelity is a prerequisite for meaningful engagement-based evaluation, and that PARASTUDENT’s simulated engagement scores support pre-deployment

trriage of AI tutor feedback, with our best variant achieving AUCs of approximately 0.80 for both feedback relevance and successful uptake, substantially above the 0.51–0.58 achieved by prompted baselines for successful uptake.

2 Related Work

2.1 LLM-based User Simulation

There is a growing body of work using LLMs to simulate human behavior (Piao et al., 2025; Manning et al., 2024; Park et al., 2023; Horton, 2023; Argyle et al., 2023; Park et al., 2024; Suh et al., 2025; Wang et al., 2026). Existing applications include psychological measurement through interaction with LLM agents (Yang et al., 2024), large-scale alignment of simulated behavior with expert benchmarks (Wang et al., 2025), and user modeling through inferred beliefs and preferences (Ryan et al., 2025; Singh et al., 2025). Another major application is the training and evaluation of LLM agents, including imagined dialogue with reinforcement learning (Hong et al., 2023), proactive and personalized training (Sun et al., 2026), recommender system evaluation (Bougie and Watanabe, 2025), and modeling language behavior (Naous et al., 2026).

Despite their popularity, recent work has highlighted the realism gap in LLM-based simulators and argued that evaluation should move beyond surface-level similarity toward distributional validation and downstream utility on real users (Suh et al., 2026; Zhou et al., 2026; Mehri et al., 2026). Our work follows this direction by first grounding PARASTUDENT in real student revision traces, then demonstrating its utility through engagement-fidelity and decision-consistency analyses.

2.2 Student Simulation

Building on recent advances, LLM-based simulation has been applied to education in settings such as classroom-level simulation (Xu et al., 2025; Mannekote et al., 2025; Zhang et al., 2025b), pedagogy and content design (Aguilar and Kang, 2023; Li et al., 2026), pre-service teacher training (Markel et al., 2023; Sanyal et al., 2025), using simulated student profiles to evaluate assessment quality (Lu and Wang, 2024; Säuberli et al., 2025), and AI tutor optimization (Dinucu-Jianu et al., 2025). Existing approaches for LLM-based student simulation have primarily relied on prompting (Lu

¹We release our code at <https://github.com/rosensh/parastudent>.

and Wang, 2024; Sonkar et al., 2024; Leinonen et al., 2025; MacNeil et al., 2024), with more recent work exploring fine-tuning approaches (Ross et al., 2026; Ross and Andreas, 2025; Scarlatos et al., 2026).

2.3 AI Tutor Evaluation

AI tutoring builds on a long tradition of Intelligent Tutoring Systems (ITS) research. Examples include studies of instructional assistance (Koedinger and Alevan, 2007), tutorial dialogue (Graesser et al., 2005), programming feedback (Fossati et al., 2015), and optimization of tutorial tactics (Chi et al., 2011). This broader literature examines how tutoring supports student learning, beyond the correctness of generated answers.

For LLM-based tutors, recent benchmarks assess pedagogical capabilities (Zhou et al., 2025; Yang et al., 2025b; Shi et al., 2026). MathTutorBench (Macina et al., 2025) and TutorBench (Srinivasa et al., 2025) evaluate teaching capabilities and core tutoring skills, while MRBench (Maurya et al., 2025) uses human-annotated dialogues to assess pedagogical quality. TutorGym (Weitekamp et al., 2025) extends this line to interactive evaluation, but still evaluates tutor actions against predefined tutor policies rather than measuring whether students actually engage with tutor feedback.

We complement these approaches by focusing on students’ uptake of specific tutor feedback: whether they ignore it, apply it incorrectly, or incorporate it successfully. This focus follows research emphasizing learners’ interpretation and use of feedback (Winstone et al., 2017; Carless and Boud, 2018). Since collecting real student responses involves logistical, ethical, and financial challenges (Yuan et al., 2026), we investigate whether PARASTUDENT can approximate these responses through feedback-conditioned code revision simulations. Our aim is to support pre-deployment triage of AI tutor feedback, complementing pedagogical evaluation and subsequent validation with real students.

3 Data

We study student code generation in the context of an introductory programming course at the University of California, Berkeley, with an average enrollment of around 900 students per semester. The course covers topics such as functions, recursion, sequences, trees, linked lists, and object-oriented

programming. Students complete approximately 10 homework assignments per semester, each with 3–6 problems. Students have access to an autograder system while they complete these assignments that provides immediate feedback on passed and failed test cases with no hidden tests. Students can keep resubmitting without penalty until their code passes all test cases. As a result, each student–problem pair has a stream of submissions.

In addition to autograder feedback, our dataset includes feedback from an AI tutor (Zamfirescu-Pereira et al., 2025), an LLM-based “Get Help” assistant deployed in the course to meet common debugging-support needs. The tutor constructs its request from the problem statement, the student’s code, and the autograder output. Following typical tutoring patterns (Markel and Guo, 2021), its prompt asks the model to assess whether the student understands the problem, which concepts need reinforcement, and whether they have a workable plan, and then to provide conceptual, debugging, or planning support as appropriate. The prompt further instructs the model to keep suggestions short and to withhold the answer and any code. The resulting feedback varies in directiveness, from conceptual scaffolding and goal reminders to concrete error diagnosis and debugging suggestions. Feedback is generated whenever a submission fails at least one test case. All submission attempts are logged, including the student’s code, autograder output, and tutor feedback. We focus on assignments implemented in Python.

Our data covers two semesters, Spring 2024 and Spring 2025, restricted to the first five Python homework assignments. We partition by semester: Spring 2025 serves as the training split (445 students, 2,022 submission streams, 6,911 submissions) and Spring 2024 as the test split (479 students, 1,546 submission streams, 6,316 submissions). We hold out the earlier semester because Spring 2024 includes more problems absent from Spring 2025, yielding a larger set of unseen problems for evaluating generalization; the two semesters use comparable assignments and the same deployed AI tutor. Appendix A gives further detail on IRB compliance and data privacy.

To evaluate generalization, we define two test subsets: **test_OP (Old Problems)** comprises 5,262 submissions from Spring 2024 students on problems that also appear in the training split, testing generalization to unseen students on familiar problems. **test_NP (New Problems)** comprises

1,054 submissions from Spring 2024 students on problems absent from the training data, testing generalization to both unseen students and unseen problems. Alongside aggregate results, we present detailed analyses and visualizations for one representative problem from each subset: test_OP_1 (1,539 submissions) and test_NP_1 (695 submissions). Results for two additional problems, test_OP_2 (968 submissions) and test_NP_2 (359 submissions), are provided in [Appendix C](#).

4 Methodology

4.1 Experiments

We study the problem of student code generation as follows: given a student s_i (the i -th student) and a programming problem p_u (the u -th problem), the model must generate the next code submission conditioned on the student’s prior submission history and, when available, the corresponding natural-language feedback instances from an AI tutor. The model sees up to k prior submissions ($k \leq 10$) with or without feedback context, depending on the experiment (we limit the prior submissions to account for the model’s context length). Each student–problem pair is represented by a stream of sequential code submissions, from the first to the final attempt.

We evaluate the LLM’s ability to generate code under two settings that separate general student-code simulation from feedback-conditioned revision modeling.

Experiment 1 (without feedback). Generate the code submission at timestamp t for student s_i on problem p_u , conditioned on the student’s prior k code submissions for the same problem:

$$c_{t,s_i,p_u} \mid (p_u, [c_{t-j,s_i,p_u}]_{j=1\dots k})$$

Experiment 2 (with feedback). Additionally, condition on the corresponding AI tutor feedback associated with each prior submission:

$$c_{t,s_i,p_u} \mid (p_u, [c_{t-j,s_i,p_u}, f_{t-j,s_i,p_u}]_{j=1\dots k})$$

where f_{t-j,s_i,p_u} denotes the feedback message linked to submission c_{t-j,s_i,p_u} .

4.2 Models

We compare supervised fine-tuning (SFT) with prompting for student code generation. We use SFT as our primary method because it directly learns from observed student revisions, aligning with

our goal of multi-dimensional behavioral imitation. Preference-based methods such as Direct Preference Optimization (DPO) ([Rafailov et al., 2023](#)) offer an alternative, but would require defining preferences that capture the multiple dimensions of student behavior we aim to reproduce. We include prompting baselines since much of prior programming student-simulation work relies on prompting, including the closest work to ours ([Leinonen et al., 2025](#)).

Fine-tuning. We SFT models using LoRA ([Hu et al., 2022](#)) ($r = 16$, $\alpha = 32$), separately for each experiment, for two epochs with a learning rate of 10^{-4} : Qwen2.5-Coder-7B ([Hui et al., 2024](#)) for its strong coding capabilities, and Qwen3-8B-Base ([Yang et al., 2025a](#)) as a complementary general-purpose model. Fine-tuning details are provided in [Appendix B](#).

Prompting. We evaluate three prompted baselines: instruction-tuned variants of our fine-tuned Qwen models (Qwen2.5-Coder-7B-Instruct and Qwen3-8B), and GPT-5 ([OpenAI, 2025a](#)) as a closed-source frontier model. Prompt templates and sampling settings are detailed in [Appendix B](#).

Throughout the paper, we refer to the five models as follows. The PARASTUDENT fine-tuned variants are qwen-parastudent (Qwen3-8B-Base) and qwen-coder-parastudent (Qwen2.5-Coder-7B). The prompted baselines are gpt-5 (GPT-5), qwen-inst (Qwen3-8B, the instruction-tuned counterpart of Qwen3-8B-Base), and qwen-coder-inst (Qwen2.5-Coder-7B-Instruct). We further conduct both fine-tuning and prompting ablations on Llama-3.1-8B and Llama-3.1-8B-Instruct ([Grattafiori et al., 2024](#)), with results reported in [Appendix C](#).

4.3 Code Fidelity Evaluation

This section presents a method for evaluating model fidelity by measuring how closely their generated code resembles real student submissions. For each student–problem instance, we generate a single next-attempt sample and evaluate it across multiple metrics grounded in educational evaluation practices.

4.3.1 Functionality Metrics

Functional correctness is the primary criterion in many programming education settings, where student submissions are commonly evaluated using autograders. We therefore evaluate each generated submission against the full test suite and com-

pute its pass rate, defined as the fraction of the problem’s autograder test cases that the submission passes. To compare simulator fidelity, we report the Wasserstein distance (WD) between real-student and model-generated pass-rate distributions and the Mean Absolute Error (MAE) between paired real and generated pass rates at the same attempt index. Lower values indicate closer alignment for both metrics.

4.3.2 Style and Code Complexity Metrics

Programming education also evaluates code beyond correctness, including readability, style, and structural complexity, taught in introductory courses (De Ruvo et al., 2018; Kirk et al., 2025). To capture whether model-generated code reflects these dimensions of student submissions, we extract:

- **PEP 8 violations:** The number of deviations from Python’s style guide (PEP 8) (van Rossum et al., 2025), serving as a proxy for stylistic cleanliness. PEP 8 violations are computed using `pycodestyle`².
- **Code length:** Measured by Lines of Code (LOC), reflecting verbosity and conciseness.
- **Abstract Syntax Tree (AST) structure:** AST depth and width (Noonan, 1985), which characterize structural complexity.

As with pass rate, we report WD between real-student and model-generated distributions and MAE between paired real and generated submissions at the same attempt index for each metric. Lower values indicate closer alignment for both measures.

4.3.3 Embedding Metrics

Programming education research has also used code embeddings to capture semantic variation in student solutions and compare patterns across learners (Azcona et al., 2019). We therefore extract 1024-dimensional code embeddings using SFR-Embedding-Code-400M (Liu et al., 2025), a lightweight model for code retrieval tasks (Li et al., 2024).

We compute two embedding-based metrics:

- **Cosine distance:** We compute $1 - \text{cosine similarity}$ between each student-written submission and the corresponding model-generated

submission for the same attempt, then average across submissions. Lower values indicate closer paired semantic alignment.

- **Top- k coverage:** For each model-generated embedding, we identify its k nearest student embeddings and report the proportion of unique student samples that appear in at least one such neighborhood. Higher values indicate broader coverage of the student code distribution. We report $k \in \{5, 10\}$.

4.4 AI Tutor Evaluation via Simulated Student Engagement

After demonstrating its code fidelity, we evaluate whether PARASTUDENT can support AI tutor feedback evaluation by measuring how simulated students respond to tutor feedback. Niousha et al. (2026) proposes *engagement scores* as a way to evaluate AI tutors beyond static pedagogical dimensions by grounding evaluation in student interactions. These scores quantify how often the student uses AI tutor feedback and, when used, how accurately it is applied across successive student submissions. We use these scores in two complementary ways. First, we measure **engagement fidelity** by comparing simulated engagement scores against real student engagement scores using score-level error metrics. Second, we evaluate **decision consistency**: whether simulator scores discriminate between high- and low-engagement diagnoses derived from real student data. This second analysis reflects a practical pre-deployment tutor-development setting, where instructors may use the simulator to inspect feedback interactions likely to receive weak or successful uptake before collecting real student interaction data.

Engagement score (Niousha et al., 2026) For a given student–problem pair (s_i, p_u) , consider the transition from submission c_{t-1, s_i, p_u} to c_{t, s_i, p_u} . Each tutor feedback message f_{t-1, s_i, p_u} consists of N_t sentences $\{\ell_{t,1}, \dots, \ell_{t,N_t}\}$, indexed by the transition $t-1 \rightarrow t$ on which they are evaluated. For each sentence $\ell_{t,m}$, $m \in \{1, \dots, N_t\}$, an LLM judge (GPT-4.1 (OpenAI, 2025b)) assigns a relevance label

$$\text{rel}(c_{t-1, s_i, p_u}, \ell_{t,m}, c_{t, s_i, p_u}) \in \{0, 1\},$$

indicating whether the sentence influenced the student’s subsequent code edit. When $\text{rel}_{t,m} = 1$, the judge also assigns a success label

$$\text{succ}(c_{t-1, s_i, p_u}, \ell_{t,m}, c_{t, s_i, p_u}) \in \{0, 1\},$$

²<https://pycodestyle.pycqa.org>

Table 1: **Fidelity of model-generated code relative to real student submissions on the aggregated test set.** WD is the population-level distance between the real-student and model-generated distributions; MAE is the student-level mean absolute error between each generated submission and the real student’s submission at the same attempt index. Lower is better for both. Bold indicates the best WD and the best MAE for each metric within each subset. Across both subsets and every metric, the best value is achieved by a PARASTUDENT variant, indicating that PARASTUDENT matches real students both distributionally and at the level of individual submissions. (Experiment 1: without AI tutor feedback; Experiment 2: with AI tutor feedback.)

(a) test_OP											
Model	Exp.	Pass Rate		PEP 8		LOC		AST Depth		AST Width	
		WD↓	MAE↓	WD↓	MAE↓	WD↓	MAE↓	WD↓	MAE↓	WD↓	MAE↓
gpt-5	1	0.597	0.610	1.300	1.685	1.974	3.522	1.013	1.347	3.290	5.733
qwen-inst	1	0.308	0.466	1.441	1.532	0.916	2.266	0.554	1.088	1.109	4.025
qwen-coder-inst	1	0.334	0.482	1.156	1.429	0.716	2.188	0.337	0.986	1.083	3.940
qwen-parastudent	1	0.052	0.283	0.125	0.640	0.121	1.441	0.260	0.779	0.393	2.991
qwen-coder-parastudent	1	0.075	0.282	0.164	0.617	0.161	1.412	0.139	0.770	0.414	2.827
gpt-5	2	0.586	0.601	1.030	1.706	2.106	3.401	0.918	1.266	3.141	5.503
qwen-inst	2	0.336	0.465	1.391	1.513	0.531	2.122	0.492	1.019	0.828	3.845
qwen-coder-inst	2	0.372	0.493	1.040	1.415	0.424	2.171	0.384	0.974	0.921	3.898
qwen-parastudent	2	0.080	0.278	0.156	0.590	0.218	1.353	0.175	0.750	0.419	2.673
qwen-coder-parastudent	2	0.083	0.279	0.200	0.613	0.204	1.390	0.154	0.756	0.432	2.778

(b) test_NP											
Model	Exp.	Pass Rate		PEP 8		LOC		AST Depth		AST Width	
		WD↓	MAE↓	WD↓	MAE↓	WD↓	MAE↓	WD↓	MAE↓	WD↓	MAE↓
gpt-5	1	0.565	0.577	1.182	1.273	1.106	2.262	1.775	1.969	4.246	5.143
qwen-inst	1	0.406	0.524	1.009	1.261	1.962	2.171	0.403	1.151	1.659	2.925
qwen-coder-inst	1	0.451	0.539	0.906	1.270	0.959	2.035	0.503	1.063	0.843	3.262
qwen-parastudent	1	0.100	0.302	0.107	0.511	0.161	0.913	0.193	0.832	0.249	1.806
qwen-coder-parastudent	1	0.026	0.296	0.077	0.515	0.253	0.995	0.151	0.842	0.254	1.841
gpt-5	2	0.562	0.576	1.166	1.246	0.283	1.973	1.090	1.379	2.309	3.588
qwen-inst	2	0.483	0.530	0.947	1.182	1.299	1.811	0.224	1.123	0.509	2.437
qwen-coder-inst	2	0.485	0.556	0.824	1.184	0.890	1.917	0.440	1.017	0.597	2.661
qwen-parastudent	2	0.007	0.286	0.089	0.477	0.219	0.881	0.146	0.779	0.238	1.665
qwen-coder-parastudent	2	0.036	0.270	0.120	0.494	0.384	0.959	0.176	0.765	0.378	1.676

indicating whether the resulting edit successfully applied the feedback. When $\text{rel}_{t,m} = 0$, the success label is undefined. We write $\text{rel}_{t,m}$ and $\text{succ}_{t,m}$ for brevity. Sentence-level labels are aggregated into RelScore and SuccScore per feedback message. RelScore measures the proportion of feedback sentences labeled as relevant ($\text{rel}=1$) among all feedback sentences, while SuccScore measures the proportion of successfully applied feedback sentences ($\text{succ}=1$) among the relevant feedback sentences. The judge also produces a natural-language rationale grounded in the code edit before assigning each label. We validate these labels against human annotations on a random sample and find moderate to substantial agreement (see Appendix D).

Inspired by Dinucu-Jianu et al. (2025), who define tutor rewards over full conversations, we compute engagement at the stream level rather than at

the level of individual feedback messages. This reflects the nature of effective feedback, which should support progress over a multi-turn revision trajectory, not merely produce local uptake after a single message (Li and Zhang, 2021). For an interaction stream with T transitions, we aggregate these message-level scores as:

$$\bar{S}_{\text{rel}} = \frac{1}{T} \sum_{t=1}^T \underbrace{\frac{1}{N_t} \sum_{m=1}^{N_t} \text{rel}_{t,m}}_{\text{RelScore}_t}.$$

Let $\mathcal{T}_{\text{rel}} = \{t : \sum_{m=1}^{N_t} \text{rel}_{t,m} > 0\}$. We define:

$$\bar{S}_{\text{succ}} = \frac{1}{|\mathcal{T}_{\text{rel}}|} \sum_{t \in \mathcal{T}_{\text{rel}}} \underbrace{\frac{\sum_{\substack{1 \leq m \leq N_t \\ \text{rel}_{t,m}=1}} \text{succ}_{t,m}}{\sum_{m=1}^{N_t} \text{rel}_{t,m}}}_{\text{SuccScore}_t}.$$

When $\mathcal{T}_{\text{rel}}$ is empty, $\bar{S}_{\text{succ}}$ is undefined, and the

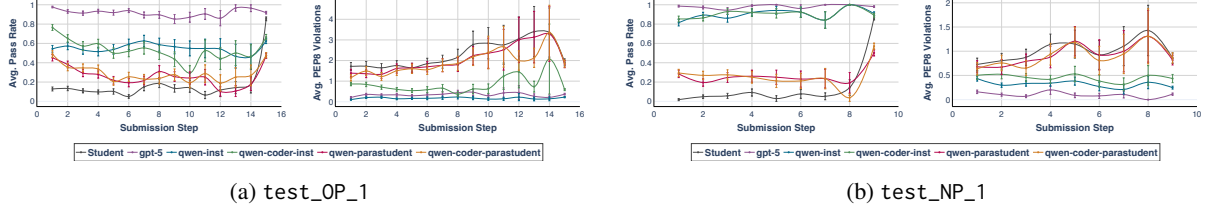


Figure 2: **Progression of pass rate (left) and PEP 8 violation count (right) across submission steps. (Experiment 2: with AI tutor feedback)** Submission steps correspond to attempt indices; final submissions and submissions at or beyond the 95th-percentile stream-length cutoff are pooled at a shared terminal step. Error bars denote the standard error of the mean over submissions contributing to each plotted point. PARASTUDENT models most closely align with the student trajectory in both metrics.

stream is excluded from analyses of successful uptake. Intuitively, $\bar{S}_{\text{rel}}$ captures how often tutor feedback is taken up in subsequent edits, while $\bar{S}_{\text{succ}}$ captures how often taken-up feedback is applied correctly across the stream.

Engagement fidelity Engagement fidelity measures how closely simulated stream-level $\bar{S}_{\text{rel}}$ and $\bar{S}_{\text{succ}}$ match the corresponding real student scores. We report MAE and RMSE to capture absolute score error.

Decision consistency Decision consistency evaluates whether simulated engagement scores distinguish streams with high versus low observed engagement. For each engagement metric, we label streams as high-engagement when their real student score exceeds the p -th percentile threshold and low otherwise, with $p \in \{25, 50\}$ to assess sensitivity to the labeling choice. The 25th-percentile threshold distinguishes relatively low-engagement cases from the remaining streams, while the 50th-percentile threshold uses the median. We evaluate continuous simulator scores using ROC-AUC, measuring their ability to discriminate between these real-student engagement labels across simulated-score cutoffs. This assesses discrimination rather than agreement at a single fixed cutoff.

5 Results

5.1 Code Fidelity

5.1.1 Aggregated Results

We evaluate code fidelity by comparing real and model-generated submissions on the aggregated test_OP and test_NP splits using the metrics in subsection 4.3.

PARASTUDENT variants consistently achieve stronger code fidelity than prompted baselines across both previously seen and unseen problems

(Table 1). Across both splits and experiments, a PARASTUDENT variant achieves the lowest WD and MAE for every metric. The advantage is particularly pronounced for pass rate, indicating substantially closer agreement with students’ functional correctness at both the population and individual-submission levels. Improvements also extend to stylistic and structural metrics.

5.1.2 Representative Problems

To illustrate how the aggregate fidelity differences appear within individual problems, we examine submission trajectories and code embeddings for two representative problems drawn from the broader test_OP and test_NP subsets: test_OP_1 (1,539 submissions) and test_NP_1 (695 submissions), respectively.

Figure 2 shows Experiment 2 submission trajectories for pass rate and PEP 8 violations. Real students begin with low pass rates, improve gradually, and show a sharp increase at the final attempt, a pattern closely matched by both PARASTUDENT variants. Prompted baselines instead achieve high pass rates from the outset, missing students’ iterative trial-and-error behavior. Stylistically, PARASTUDENT more closely reproduces the persistent PEP 8 violations in student code, whereas prompted baselines produce code with fewer violations.

Embedding-based metrics provide complementary evidence (Table 2). Across both representative problems and experiments, PARASTUDENT variants achieve lower cosine distances and higher coverage than prompted baselines, indicating closer alignment with real student code in embedding space. On test_OP_1 (Table 2a), their cosine distances range from 0.014 to 0.017, and top-10 coverage reaches 98.0%, compared with 71.1% for the best prompted baseline. The trend is similar on test_NP_1 (Table 2b). The Experiment 2

Table 2: **Embedding-based Similarity Metrics.** Cosine distance measures per-submission alignment with the real student code, and top- k coverage indicates the proportion of student codes represented in model neighborhoods (see [subsubsection 4.3.3](#) for details). Lower distances and higher coverage indicate better alignment with student code distribution. The PARASTUDENT variants consistently align most closely with the real student distribution across all metrics. Bold indicates the best result for each metric across both experiments. (Experiment 1: without AI tutor feedback, Experiment 2: with AI tutor feedback).

(a) test_OP_1					(b) test_NP_1				
Model	Exp.	Cosine Distance ↓	Coverage (%) ↑		Model	Exp.	Cosine Distance ↓	Coverage (%) ↑	
			k=5	k=10				k=5	k=10
gpt-5	1	0.035	20.5	28.7	gpt-5	1	0.053	29.5	45.6
qwen-inst	1	0.028	30.4	46.0	qwen-inst	1	0.045	36.7	54.8
qwen-coder-inst	1	0.028	56.7	71.1	qwen-coder-inst	1	0.042	60.7	74.8
qwen-parastudent	1	0.016	93.7	97.4	qwen-parastudent	1	0.021	94.7	98.1
qwen-coder-parastudent	1	0.016	94.0	98.0	qwen-coder-parastudent	1	0.022	94.1	98.1
gpt-5	2	0.029	51.1	66.8	gpt-5	2	0.043	44.5	63.7
qwen-inst	2	0.022	53.9	70.0	qwen-inst	2	0.039	46.6	64.8
qwen-coder-inst	2	0.027	55.9	70.8	qwen-coder-inst	2	0.038	60.7	75.7
qwen-parastudent	2	0.014	92.8	97.5	qwen-parastudent	2	0.018	91.4	96.1
qwen-coder-parastudent	2	0.017	92.8	97.3	qwen-coder-parastudent	2	0.021	92.5	97.6

(a) test_OP_1

(b) test_NP_1

Figure 3: **Code embedding visualization (Experiment 2: with AI tutor feedback).** Real student code (gray squares) is compared against gpt-5 (purple circles), qwen-inst (blue circles), qwen-coder-inst (green circles), qwen-parastudent (pink circles), and qwen-coder-parastudent (orange circles). 1024-dimensional embeddings are projected onto a 2D plane using Principal Component Analysis (PCA) for visualization purposes. PARASTUDENT variants show the substantially stronger overlap with the student code distribution compared to baselines, including better coverage of outlier codes where there is less clustering.

projections in [Figure 3](#) visually reinforce these findings: PARASTUDENT representations overlap more closely with real student code across both dense clusters and lower-density regions, whereas prompted baselines exhibit less overlap and narrower coverage.

5.2 Real vs. Simulated Student Engagement with the AI Tutor

We evaluate whether PARASTUDENT can support pre-deployment assessment of AI tutor feedback on the aggregated test_NP split, comparing both PARASTUDENT variants with all three prompted baselines. This held-out new-problem setting approximates evaluating feedback for a new problem before collecting real student interaction data. We focus on Experiment 2, which conditions simulations on the tutor feedback received by students and corresponds to the intended feedback-evaluation setting.

Following [subsection 4.4](#), we evaluate **engagement fidelity**, measuring alignment between simulated and real engagement scores, and **decision**

Table 3: **Simulated engagement fidelity on the aggregated test_NP split.** All models use Experiment 2 (with AI tutor feedback). Lower is better.

Score	Model	MAE ↓	RMSE ↓
$\bar{S}_{rel}$	gpt-5	0.21	0.27
$\bar{S}_{rel}$	qwen-inst	0.24	0.30
$\bar{S}_{rel}$	qwen-coder-inst	0.23	0.29
$\bar{S}_{rel}$	qwen-parastudent	0.22	0.28
$\bar{S}_{rel}$	qwen-coder-parastudent	0.13	0.21
$\bar{S}_{succ}$	gpt-5	0.32	0.43
$\bar{S}_{succ}$	qwen-inst	0.32	0.43
$\bar{S}_{succ}$	qwen-coder-inst	0.31	0.42
$\bar{S}_{succ}$	qwen-parastudent	0.24	0.37
$\bar{S}_{succ}$	qwen-coder-parastudent	0.19	0.31

consistency, testing whether simulated scores identify interaction streams with high versus low real engagement. We assess both feedback relevance ($\bar{S}_{rel}$) and successful uptake ($\bar{S}_{succ}$).

5.2.1 Engagement Fidelity

[Table 3](#) reports score-level engagement fidelity on the aggregated test_NP split under Experiment 2. MAE and RMSE quantify deviations between sim-

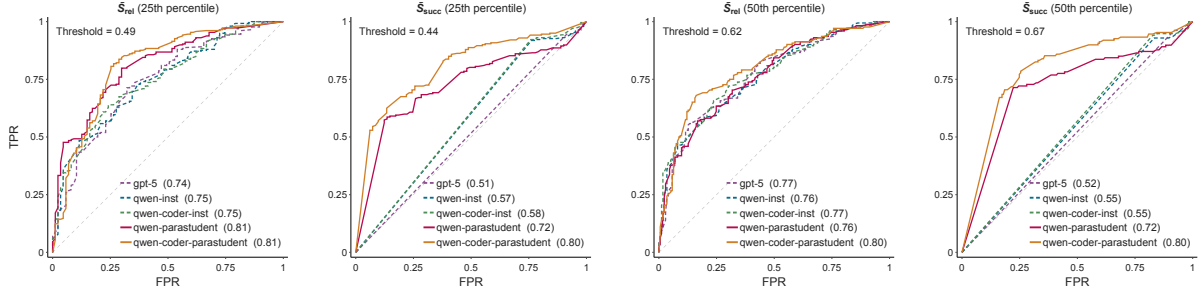


Figure 4: **Engagement ROC curves on the aggregated test_NP split.** All models use tutor feedback (Experiment 2). From left to right, panels show relevance $\bar{S}_{rel}$ and successful uptake $\bar{S}_{succ}$ at the 25th-percentile threshold, followed by the same metrics at the 50th-percentile threshold. Thresholds are computed from real engagement scores; panel annotations give threshold values, and legends report AUCs. Dashed lines denote prompted baselines; solid lines denote PARASTUDENT variants. Both variants achieve higher AUCs for successful uptake at both thresholds.

ulated and real stream-level engagement scores.

qwen-coder-parastudent achieves the lowest reported errors for both scores. For relevance $\bar{S}_{rel}$, it obtains an MAE of 0.13 and RMSE of 0.21, compared with 0.21–0.24 and 0.27–0.30, respectively, for prompted baselines. For successful uptake $\bar{S}_{succ}$, its MAE is 0.19, and RMSE is 0.31, compared with 0.31–0.32 and 0.42–0.43 for prompted baselines.

Relative to prompted baselines, both PARASTUDENT variants achieve lower estimation errors for successful uptake $\bar{S}_{succ}$, and the coder variant yields the lowest MAE and RMSE across both engagement metrics. These results indicate better agreement with observed engagement, suggesting an advantage over prompted models in approximating how students engage with tutor feedback.

5.2.2 Decision Consistency

Figure 4 shows ROC curves and AUCs on the aggregated test_NP split under Experiment 2. We evaluate whether simulated scores distinguish interaction streams whose real engagement exceeds the 25th- or 50th-percentile threshold. Corresponding 95% confidence intervals are reported in Appendix C.

The clearest separation between PARASTUDENT and prompted baselines occurs for successful uptake $\bar{S}_{succ}$. At both thresholds, qwen-coder-parastudent achieves an AUC of approximately 0.80 and qwen-parastudent achieves approximately 0.72, compared with 0.51–0.58 across prompted baselines. For relevance $\bar{S}_{rel}$, differences are smaller: the coder variant achieves AUCs of 0.81 and 0.80 at the 25th- and 50th-percentile thresholds, respectively, compared with

0.74–0.75 and 0.76–0.77 for prompted baselines.

This pattern is consistent with the code-fidelity results (subsection 5.1), where prompted baselines produce overly successful revisions and miss students’ gradual trial-and-error behavior. Under Experiment 2, all models receive AI tutor feedback and can generate edits related to it; relevance alone therefore does not establish that a simulator reproduces novice feedback uptake. Successful uptake additionally measures whether taken-up feedback is applied correctly. The stronger discrimination of PARASTUDENT for $\bar{S}_{succ}$ is thus consistent with better modeling of students’ imperfect application of feedback.

Practically, these findings suggest a role for PARASTUDENT in pre-deployment triage of AI tutor feedback. Simulated engagement scores can help prioritize feedback for instructor review, while generated revision trajectories and sentence-level uptake labels illustrate how students might respond. Instructors can inspect simulated cases of feedback being ignored, applied incorrectly, or successfully incorporated, and use these examples to evaluate and refine AI tutor feedback.

6 Conclusion

We introduced PARASTUDENT for pre-deployment AI tutor evaluation. By fine-tuning on real revision trajectories, it matches real student code more closely than prompted baselines across functional, stylistic, and semantic dimensions, and its simulated engagement scores better discriminate high-versus low-uptake streams. Together, these findings show how simulated engagement can support pre-deployment triage of AI tutor feedback.

Limitations

While our findings highlight the importance of fine-tuning and holistic evaluation of LLMs in student simulation, several limitations must be acknowledged.

- This work is limited to one course (with a large enrollment of around 900 students). The training and test datasets are drawn from different semesters of the same course. Obtaining a comparable longitudinal dataset with iterative student submission and associated AI tutor feedback is particularly challenging in the education domain. We note, however, that our test_NP setting — where the simulator is evaluated on problems unseen during training — provides a partial signal of generalizability, since it requires the model to simulate student behavior on novel problem structures without problem-specific fine-tuning. Extending evaluation to other courses, institutions, and programming languages is an important next step.
- A more comprehensive comparison across a broader range of model types, sizes, and families is an important direction for future research. Nevertheless, these baselines are representative of commonly used instruction-tuned and frontier models, and the consistent trends observed across them support our conclusions about the limitations of prompt-based student simulation. The consistent advantage of fine-tuning over prompting across both model families tested (Qwen and Llama) suggests this finding is unlikely to be model-specific, though verification on additional architectures remains open.

Potential Risks. While simulating student code can offer pedagogical benefits, it also raises several risks. First, if misapplied, such models could reinforce incorrect programming habits or misconceptions by overfitting to common student errors. Additionally, any deployment of such systems in educational settings must be done with care, including appropriate safeguards for data anonymization and ethical use.

References

Jair J Aguilar and Seokmin Kang. 2023. Innovating with in-service mathematics teachers' profes-

sional development: The intersection among mixed-reality simulations, approximation-of-practice, and technology-acceptance. *International Electronic Journal of Mathematics Education*, 18(4):em0750.

Lisa P Argyle, Ethan C Busby, Nancy Fulda, Joshua R Gubler, Christopher Rytting, and David Wingate. 2023. Out of one, many: Using language models to simulate human samples. *Political Analysis*, 31(3):337–351.

David Azcona, Piyush Arora, I-Han Hsiao, and Alan Smeaton. 2019. user2code2vec: Embeddings for profiling students based on distributional representations of source code. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, pages 86–95.

Nicolas Bougie and Narimawa Watanabe. 2025. Simuser: Simulating user behavior with large language models for recommender system evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 43–60.

David Carless and David Boud. 2018. The development of student feedback literacy: Enabling uptake of feedback. *Assessment & evaluation in higher education*, 43(8):1315–1325.

Min Chi, Kurt VanLehn, Diane Litman, and Pamela Jordan. 2011. An evaluation of pedagogical tutorial tactics for a natural language tutoring system: A reinforcement learning approach. *International Journal of Artificial Intelligence in Education*, 21(1-2):83–113.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

Giuseppe De Ruvo, Ewan Tempero, Andrew Luxton-Reilly, Gerard B Rowe, and Nasser Giacaman. 2018. Understanding semantic style by analysing student code. In *Proceedings of the 20th Australasian Computing Education Conference*, pages 73–82.

David Dinucu-Jianu, Jakub Macina, Nico Daheim, Ido Hakimi, Iryna Gurevych, and Mrinmaya Sachan. 2025. From problem-solving to teaching problem-solving: Aligning llms with pedagogy using reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 272–292.

Soukaina Ezzaki, Najat Messaoudi, and Jaafar Khalid Naciri. 2025. Evaluating educational content through virtual student simulations using large language models. In *International Symposium on Generative AI and Education*, pages 241–250. Springer.

Davide Fossati, Barbara Di Eugenio, STELLAN Ohlsson, Christopher Brown, and Lin Chen. 2015. Data driven automatic feedback generation in the ilist intelligent tutoring system. *Technology, Instruction, Cognition and Learning*, 10(1):5–26.

- Arthur C Graesser, Patrick Chipman, Brian C Haynes, and Andrew Olney. 2005. Autotutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions on Education*, 48(4):612–618.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Joey Hong, Sergey Levine, and Anca Dragan. 2023. Zero-shot goal-directed dialogue via rl on imagined conversations. *arXiv preprint arXiv:2311.05584*.
- John J Horton. 2023. Large language models as simulated economic agents: What can we learn from homo silicus? Technical report, National Bureau of Economic Research.
- Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. [LoRA: Low-rank adaptation of large language models](#). In *International Conference on Learning Representations*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Hyoungwook Jin, Minju Yoo, Jeongeon Park, Yokyung Lee, Xu Wang, and Juho Kim. 2025. Teachtune: Reviewing pedagogical agents against diverse student profiles with simulated students. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–28.
- Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems*, pages 1–20.
- Diana Kirk, Andrew Luxton-Reilly, and Ewan Tempero. 2025. Distilling pep 8 for teaching introductory programming. In *Proceedings of the 27th Australasian Computing Education Conference*, pages 26–35.
- Kenneth R Koedinger and Vincent Aleven. 2007. Exploring the assistance dilemma in experiments with cognitive tutors. *Educational psychology review*, 19(3):239–264.
- Juho Leinonen, Paul Denny, Olli Kiljunen, Stephen MacNeil, Sami Sarsa, and Arto Hellas. 2025. [Llm-itation is the sincerest form of data: Generating synthetic buggy code submissions for computing education](#). In *Proceedings of the 27th Australasian Computing Education Conference, ACE '25*, page 56–63. Association for Computing Machinery.
- Haoyuan Li, Jifan Yu, Xin Cong, Yang Dang, Daniel Zhang-Li, Lu Mi, Yisi Zhan, Huiqin Liu, and Zhiyuan Liu. 2026. Which type of students can llms act? investigating authentic simulation with graph-based human-ai collaborative system. *AI Open*.
- Wentao Li and Fuhui Zhang. 2021. Tracing the path toward self-regulated revision: An interplay of instructor feedback, peer feedback, and revision goals. *Frontiers in Psychology*, 11:612088.
- Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Yichun Yin, Hao Zhang, Yong Liu, Yasheng Wang, and Ruiming Tang. 2024. [Coir: A comprehensive benchmark for code information retrieval models](#). *CoRR*, abs/2407.02883.
- Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. 2025. Beyond single-turn: A survey on multi-turn interactions with large language models. *arXiv preprint arXiv:2504.04717*.
- Q Vera Liao and Ziang Xiao. 2023. Rethinking model evaluation as narrowing the socio-technical gap. *arXiv preprint arXiv:2306.03100*.
- Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 2024. Teaching cs50 with ai: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM technical symposium on computer science education V. 1*, pages 750–756.
- Ye Liu, Rui Meng, Shafiq Joty, Silvio Savarese, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2025. [CodeXEmbed: A generalist embedding model family for multilingual and multi-task code retrieval](#). In *Second Conference on Language Modeling*.
- Xinyi Lu and Xu Wang. 2024. Generative students: Using llm-simulated student profiles to support question item evaluation. In *Proceedings of the Eleventh ACM conference on learning@ Scale*, pages 16–27.
- Jakub Macina, Nico Daheim, Ido Hakimi, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. 2025. Math-tutorbench: A benchmark for measuring open-ended pedagogical capabilities of llm tutors. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 204–221.
- Stephen MacNeil, Magdalena Rogalska, Juho Leinonen, Paul Denny, Arto Hellas, and Xandria Crosland. 2024. [Synthetic students: A comparative study of bug distribution between large language models and computing students](#). In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1, SIGCSE Virtual 2024*, page 137–143. Association for Computing Machinery.
- Amogh Mannekote, Adam Davies, Jina Kang, and Kristy Elizabeth Boyer. 2025. [Can llms reliably simulate human learner actions? a simulation authoring framework for open-ended learning environments](#). *Proceedings of the AAAI Conference on Artificial Intelligence*.

- Benjamin S Manning, Kehang Zhu, and John J Horton. 2024. Automated social science: Language models as scientist and subjects. Technical report, National Bureau of Economic Research.
- Julia M Markel and Philip J Guo. 2021. Inside the mind of a cs undergraduate ta: A firsthand account of undergraduate peer tutoring in computer labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 502–508.
- Julia M Markel, Steven G Opferman, James A Landay, and Chris Piech. 2023. Gpteach: Interactive ta training with gpt-based students. In *Proceedings of the tenth acm conference on learning@ scale*, pages 226–236.
- Kaushal Kumar Maurya, Kv Aditya Srivatsa, Kseniia Petukhova, and Ekaterina Kochmar. 2025. [Unifying AI tutor evaluation: An evaluation taxonomy for pedagogical ability assessment of LLM-powered AI tutors](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1234–1251, Albuquerque, New Mexico. Association for Computational Linguistics.
- Shuhaib Mehri, Philippe Laban, Sumuk Shashidhar, Marwa Abdulhai, Sergey Levine, Michel Galley, and Dilek Hakkani-Tür. 2026. Measuring and mitigating the distributional gap between real and simulated user behaviors. *arXiv preprint arXiv:2605.07847*.
- Sheshera Mysore, Debarati Das, Hancheng Cao, and Bahareh Sarrafzadeh. 2025. Prototypical human-ai collaboration behaviors from llm-assisted writing in the wild. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16830–16857.
- Tarek Naous, Philippe Laban, Wei Xu, and Jennifer Neville. 2026. [Flipping the dialogue: Training and evaluating user language models](#). In *International Conference on Learning Representations*.
- Rose Niousha, Samantha Boatright Smith, Bitu Akram, Peter Brusilovsky, Arto Hellas, Juho Leinonen, John DeNero, and Narges Norouzi. 2026. The missing evaluation axis: What 10,000 student submissions reveal about ai tutor effectiveness. In *International Conference on Artificial Intelligence in Education*, pages 384–398. Springer.
- Robert E. Noonan. 1985. [An algorithm for generating abstract syntax trees](#). *Comput. Lang.*, 10(3–4):225–236.
- OpenAI. 2025a. [Gpt-5 is here](#).
- OpenAI. 2025b. [Introducing gpt-4.1 in the api](#).
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. [Generative agents: Interactive simulacra of human behavior](#). In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.
- Joon Sung Park, Carolyn Q. Zou, Aaron Shaw, Benjamin Mako Hill, Carrie Cai, Meredith Ringel Morris, Robb Willer, Percy Liang, and Michael S. Bernstein. 2024. [Generative agent simulations of 1,000 people](#).
- Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, and 1 others. 2025. Agentsocty: Large-scale simulation of llm-driven generative agents advances understanding of human behaviors and society. *arXiv preprint arXiv:2502.08691*.
- Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, and 1 others. 2025. Userbench: An interactive gym environment for user-centric agents. *arXiv preprint arXiv:2507.22034*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.
- Alexis Ross and Jacob Andreas. 2025. [Learning to make mistakes: Modeling incorrect student thinking and key errors](#).
- Alexis Ross, Megha Srivastava, Jeremiah Blanchard, and Jacob Andreas. 2026. Modeling student learning with 3.8 million program traces. In *International Conference on Artificial Intelligence in Education*, pages 324–338. Springer.
- Michael J. Ryan, Omar Shaikh, Aditri Bhagirath, Daniel Frees, William Held, and Diyi Yang. 2025. [SynthesizeMe! inducing persona-guided prompts for personalized reward models in LLMs](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.
- Debdeep Sanyal, Agniva Maiti, Umakanta Maharana, Dhruv Kumar, Ankur Mali, C Lee Giles, and Murari Mandal. 2025. Investigating pedagogical teacher and student llm agents: Genetic adaptation meets retrieval-augmented generation across learning styles. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13359–13400.
- Andreas Säuberli, Diego Frassinelli, and Barbara Plank. 2025. Do llms give psychometrically plausible responses in educational assessments? In *Proceedings of the 20th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2025)*, pages 266–278.
- Alexander Scarlatos, Jaewook Lee, Simon Woodhead, and Andrew Lan. 2026. Simulated students in tutoring dialogues: Substance or illusion? In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 42349–42385.

- Weikang Shi, Houxing Ren, Junting Pan, Aojun Zhou, Ke Wang, Zimu Lu, Yunqiao Yang, Yuxuan Hu, Linda Wei, Mingjie Zhan, and 1 others. 2026. From solver to tutor: Evaluating the pedagogical intelligence of llms with kmp-bench. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 32965–32973.
- Anikait Singh, Sheryl Hsu, Kyle Hsu, Eric Mitchell, Stefano Ermon, Tatsunori Hashimoto, Archit Sharma, and Chelsea Finn. 2025. [FSPO: Few-shot preference optimization of synthetic preference data elicits LLM personalization to real users](#). In *2nd Workshop on Models of Human Feedback for AI Alignment*.
- Shashank Sonkar, Xinghe Chen, Naiming Liu, Richard G Baraniuk, and Mrinmaya Sachan. 2024. Llm-based cognitive models of students with misconceptions. *arXiv preprint arXiv:2410.12294*.
- Rakshith S Srinivasa, Zora Che, Chen Bo Calvin Zhang, Diego Mares, Ernesto Hernandez, Jayeon Park, Dean Lee, Guillermo Mangialardi, Charmaine Ng, Ed-Yeremai Hernandez Cardona, and 1 others. 2025. Tutorbench: A benchmark to assess tutoring capabilities of large language models. *arXiv preprint arXiv:2510.02663*.
- Joseph Suh, Erfan Jahanparast, Suhong Moon, Minwoo Kang, and Serina Chang. 2025. Language model fine-tuning on scaled survey data for predicting distributions of public opinions. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21147–21170.
- Joseph Suh, Ayush Raj, Minwoo Kang, and Serina Chang. 2026. Quantifying the utility of user simulators for building collaborative llm assistants. *arXiv preprint arXiv:2605.09808*.
- Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. 2026. [Training proactive and personalized LLM agents](#). In *Third Conference on Language Modeling*.
- Guido van Rossum, Barry Warsaw, and Alyssa Coghlan. 2025. [Pep 8 – style guide for python code](#).
- Qihan Wang, Nicholas Tomlin, Michael Hu, Brian Dillon, and Tal Linzen. 2026. Simulating human memory with language models. *arXiv preprint arXiv:2605.25680*.
- Yunzhe Wang, Gale Lucas, Burcin Becerik-Gerber, and Volkan Ustun. 2025. Implicit behavioral alignment of language agents in high-stakes crowd simulations. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 30669–30686.
- Daniel Weitekamp, Momin N. Siddiqui, and Christopher J. MacLellan. 2025. Tutorgym: A testbed for evaluating ai agents as tutors and students. In *International Conference on Artificial Intelligence in Education*, pages 361–376. Springer.
- Naomi E Winstone, Robert A Nash, Michael Parker, and James Rowntree. 2017. Supporting learners’ agentic engagement with feedback: A systematic review and a taxonomy of recipience processes. *Educational psychologist*, 52(1):17–37.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. [Hugging-face’s transformers: State-of-the-art natural language processing](#). *Preprint*, arXiv:1910.03771.
- Songlin Xu, Hao-Ning Wen, Hongyi Pan, Dallas Dominguez, Dongyin Hu, and Xinyu Zhang. 2025. [Classroom simulacra: Building contextual student generative agents in online education for learning behavioral simulation](#). In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Qisen Yang, Zekun Wang, Honghui Chen, Shenzhi Wang, Yifan Pu, Xin Gao, Wenhao Huang, Shiji Song, and Gao Huang. 2024. Psychogat: A novel psychological measurement paradigm through interactive fiction games with llm agents. *arXiv preprint arXiv:2402.12326*.
- Tengchao Yang, Sichen Guo, Mengzhao Jia, Jiaming Su, Yuanyang Liu, Zhihan Zhang, and Meng Jiang. 2025b. Mmtutorbench: The first multimodal benchmark for ai math tutoring. *arXiv preprint arXiv:2510.23477*.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. 2025. [\$\tau\$ -bench: A benchmark for Tool-Agent-User interaction in real-world domains](#). In *International Conference on Learning Representations*.
- Zhihao Yuan, Yunze Xiao, Ming Li, Weihao Xuan, Richard Tong, Mona Diab, and Tom Mitchell. 2026. Towards valid student simulation with large language models. *arXiv preprint arXiv:2601.05473*.
- JD Zamfirescu-Pereira, Laryn Qi, Björn Hartmann, John DeNero, and Narges Norouzi. 2025. 61a bot report: Ai assistants in cs1 save students homework time and reduce demands on staff.(now what?). In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 1309–1315.
- Erhan Zhang, Xingzhu Wang, Peiyuan Gong, Zixuan Yang, and Jiaxin Mao. 2025a. Exploring human-like thinking in search simulations with large language models. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2669–2673.

Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhanxin Hao, Jianxiao Jiang, Jie Cao, Huiqin Liu, Zhiyuan Liu, and 1 others. 2025b. Simulating classroom education with llm-empowered agents. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 10364–10379.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhonghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Chengliang Zhou, Mei Wang, Ting Zhang, Qiannan Zhu, Jian Li, and Hua Huang. 2025. From answers to questions: Eqgbench for evaluating llms’ educational question generation. *arXiv preprint arXiv:2508.10005*.

Xuhui Zhou, Weiwei Sun, Qianou Ma, Yiqing Xie, Jiarui Liu, Weihua Du, Sean Welleck, Yiming Yang, Graham Neubig, Tongshuang Wu, and Maarten Sap. 2026. [Mind the Sim2Real gap in user simulation for agentic tasks](#). In *Third Conference on Language Modeling*.

A Data

Privacy. We received IRB approval for the research use of the dataset (protocol ID: 2023-09-16725), after which the AI tutor was deployed in the course. When students submitted their code to the auto-grader as part of the regular course practice, they were asked whether they would like to receive AI tutor feedback on their code, and if so, were asked whether they would like their data to be used for research. Students were still able to receive AI feedback even if they did not wish their data to be used in research. Only students who explicitly consented were included in our analysis. Additionally, all student identifiers (e.g., IDs and emails) were fully anonymized prior to passing the data to the LLM and any further analysis. As part of our system design, any data from LLM calls from our educational platform to the model providers will not be retained on third-party servers and will not be used for further fine-tuning of those models.

Documentation. Data contains student code submissions for an introductory programming course from the University of California, Berkeley. We filter data from the Spring 2024 and Spring 2025 semesters. Additionally, we filter only assignment problems in Python. Student demographic

information is not available due to privacy regulations.

B Models

Models. We used the following models for fine-tuning experiments: Qwen2.5-Coder-7B (Hui et al., 2024), Qwen3-8B-Base (Yang et al., 2025a), and Llama-3.1-8B (Grattafiori et al., 2024). We used the following models for prompting experiments: Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), Qwen3-8B (Yang et al., 2025a), GPT-5 (OpenAI, 2025a), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024).

Infrastructure. All fine-tuning experiments were run on a single Standard NC40ads H100 v5 (40 vcpus, 320 GiB memory) GPU on Microsoft Azure. Across all experiments, including model ablations, the total compute usage was 12.54 GPU hours. We used the same machine for data-generation experiments with open-source models. We used Azure’s OpenAI API to sample from GPT-5.

B.1 Fine-tuning

Due to the large number of experiments and model variants, we used Low-Rank Adaptation (LoRA) (Hu et al., 2022) for parameter-efficient fine-tuning. We used the following commonly used parameters: rank $r=16$, scaling factor $\alpha=32$, and a dropout rate of 0.05. LoRA adapters were applied to all linear layers. Models were fine-tuned for two epochs with a batch size of 16, a learning rate of 10^{-4} , a cosine learning rate scheduler, and the AdamW optimizer. All experiments were conducted using bfloat16 precision (bf16=True). HuggingFace’s (Wolf et al., 2020) transformers and peft libraries were used for fine-tuning.

For sampling from fine-tuned models, we followed common practice and used the following sampling parameters: temperature=0.7, top-p=0.8, top-k=20, and min-p=0.0. Prompt templates are shown in Figure 5; the same prompt templates were used for formatting the training data. All code snippets in training and sampling data (including skeleton code snippets) are wrapped in `<code>` and `</code>`.

B.2 Prompting

For sampling from prompt-based models, we followed common practice and used the following parameters: (1) Qwen2.5-Coder-7B-Instruct, Qwen3-8B, and Llama-3.1-8B-Instruct (temperature=0.7,

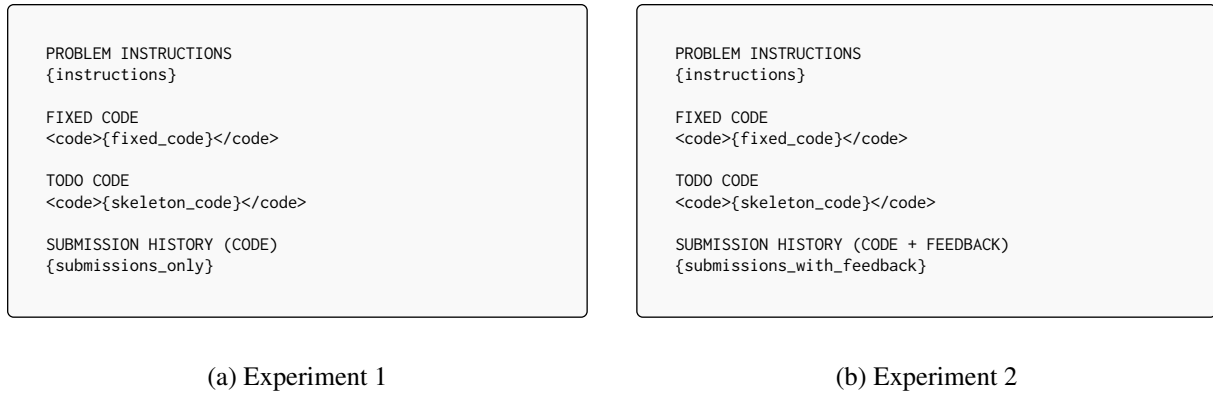


Figure 5: **Prompt templates for fine-tuned models.** Left (Experiment 1 without feedback), Right (Experiment 2 with feedback).

top-p=0.8, top-k=20, and min-p=0.0), (2) GPT-5 (temperature=1.0, top-p=1.0). We used the following prompt for all prompted models as shown in Figure 6.

C Results

This section provides extended results complementing the main paper. We organize the following sections:

- **Additional problem sets.** We report results on test_OP_2 and test_NP_2 to evaluate model performance on additional sets of test problems. Functionality and style results are shown in Figure 7, embedding-based metrics in Table 4 and Figure 8. Across these additional problems, we observe patterns consistent with test_OP_1 and test_NP_1: PARASTUDENT variants consistently achieve closer alignment with real student code than prompt-based baselines.
- **Detailed engagement AUCs.** Table 6 reports ROC AUCs with 95% confidence intervals for all models on the aggregated test_NP split under Experiment 2, using the 25th- and 50th-percentile thresholds of real engagement scores.
- **Fine-tuning ablation.** We report Wasserstein distance results for the Llama-3.1-8B ablation (Table 5). The trend echoes that of the Qwen models: the fine-tuned llama-parastudent achieves substantially closer distributional alignment with real student code than its instruction-tuned counterpart llama-inst across all metrics.

D LLM Judge for Feedback Engagement

D.1 Prompt

Figure 9 shows the LLM judge prompt used to label feedback sentence engagement using GPT-4.1 (OpenAI, 2025b). The prompt provides the problem context, full tutor feedback, a target feedback sentence, successive student code submissions, and a code diff summary, and tasks the judge with determining whether the sentence was relevant to the code edits and whether the resulting changes were correct.

D.2 Human Validation Instruction to Annotators

Two annotators reviewed 50 randomly sampled feedback-code edit pairs to validate the labels produced by the LLM judge. For each example, annotators were shown the full problem context, the complete AI tutor feedback, a target feedback sentence, successive student code submissions, and a code difference summary highlighting the changes between submissions. Their task was to assess whether the target feedback sentence was relevant to the student’s code edits and, if so, whether the resulting edits were applied correctly.

Specifically, annotators provided the following labels:

- **is_relevant:** true if the student’s code edits were directly motivated by the feedback sentence, and false otherwise.
- **is_success:** true if the edit fixed or improved the targeted issue; false if the edit was incorrect or introduced new errors; and null if the feedback sentence was not relevant to the edit (is_relevant = false).

System message YOUR TASK You are simulating a student taking an introduction to Python programming course. Generate the student's next attempt at the TODO CODE of the current problem, conditioned on their previous code submissions. * Only write the contents of the TODO CODE. Do not include docstrings or anything outside it. * As a novice student, your attempts may include mistakes or partial fixes. * FIXED CODE is provided; do not copy, modify, or include it. * Wrap the generated code in <code><code></code> and <code></code></code> tags so that it can be easily extracted. * If the generated attempt is the student's final submission, append "<code><SUBMIT></code>" at the end. * Use Python programming language User message PROBLEM INSTRUCTIONS {instructions} FIXED CODE <code>{fixed_code}</code> TODO CODE <code>{skeleton_code}</code> SUBMISSION HISTORY (CODE) {submissions_only}	System message YOUR TASK You are simulating a student taking an introduction to Python programming course. Generate the student's next attempt at the TODO CODE of the current problem, using both their prior submissions and the feedback they received. * Only write the contents of the TODO CODE. Do not include docstrings or anything outside it. * As a novice student, your attempts may include mistakes or partial fixes. * FIXED CODE is provided; do not copy, modify, or include it. * Use prior feedback to inform your code revision. * Wrap the generated code in <code><code></code> and <code></code></code> tags so that it can be easily extracted. * If the generated attempt is the student's final submission, append "<code><SUBMIT></code>" at the end. * Use Python programming language User message PROBLEM INSTRUCTIONS {instructions} FIXED CODE <code>{fixed_code}</code> TODO CODE <code>{skeleton_code}</code> SUBMISSION HISTORY (CODE + FEEDBACK) {submissions_with_feedback}
---	--

(a) Experiment 1

(b) Experiment 2

Figure 6: **Prompt templates for prompt-based models.** Left (Experiment 1 without feedback), Right (Experiment 2 with feedback). These prompts are adapted from prior work (Leinonen et al., 2025), which used LLMs to mimic the distribution of error types and test case failure frequencies of real student datasets.

Both annotators were authors of the paper and were familiar with the task setting and course content. Annotators were informed that all examples consisted of anonymized student code and AI-generated feedback from an introductory programming course and contained no personally identifiable information. The task involved minimal risk, and annotators could stop at any time. No monetary compensation was provided. The annotation interface consisted of a spreadsheet with predefined fields for the labels.

D.3 LLM-Human Agreement

We assessed the reliability of the LLM judge using 50 randomly sampled feedback–code edit pairs annotated independently by two human annotators. Because these labels are highly context-dependent, human annotation is time-consuming, which motivated a targeted validation rather than a large-scale labeling effort. We report pairwise agreement between the LLM judge and each human annotator, as well as human–human agreement, using Cohen’s κ (Cohen, 1960). As shown in Table 7, agreement

was moderate to substantial across both *Relevant* and *Success* labels.

E AI Assistance

We used AI for writing assistance and code, including but not limited to Codex and Claude Code.

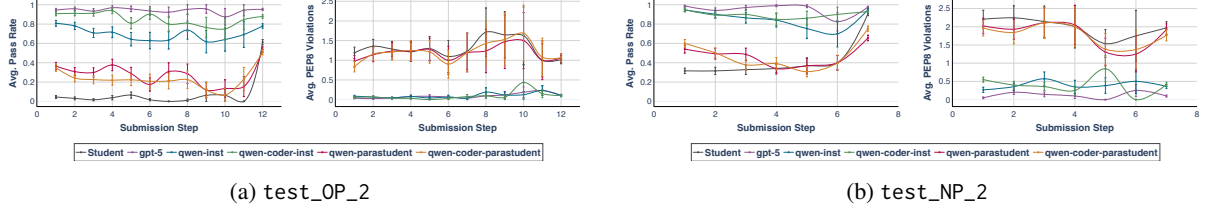


Figure 7: Progression of pass rate (left) and PEP 8 violation count (right) across submission steps on test_OP_2 and test_NP_2 (Experiment 2: with AI tutor feedback).

Table 4: Embedding-based Similarity Metrics on test_OP_2 and test_NP_2. (Experiment 1: without AI tutor feedback, Experiment 2: with AI tutor feedback)

(a) test_OP_2					(b) test_NP_2				
Model	Exp.	Cosine Distance ↓	Coverage (%) ↑		Model	Exp.	Cosine Distance ↓	Coverage (%) ↑	
			k=5	k=10				k=5	k=10
gpt-5	1	0.056	40.1	53.5	gpt-5	1	0.041	20.3	30.4
qwen-inst	1	0.078	32.2	46.4	qwen-inst	1	0.025	37.3	45.4
qwen-coder-inst	1	0.082	33.2	49.5	qwen-coder-inst	1	0.024	57.1	72.1
qwen-parastudent	1	0.025	94.8	97.8	qwen-parastudent	1	0.016	94.7	97.8
qwen-coder-parastudent	1	0.023	94.3	97.7	qwen-coder-parastudent	1	0.015	96.1	98.3
gpt-5	2	0.043	59.8	73.5	gpt-5	2	0.034	49.0	56.0
qwen-inst	2	0.057	57.0	72.0	qwen-inst	2	0.027	52.4	62.4
qwen-coder-inst	2	0.064	52.9	70.6	qwen-coder-inst	2	0.025	59.1	69.6
qwen-parastudent	2	0.024	92.8	96.1	qwen-parastudent	2	0.014	93.6	96.4
qwen-coder-parastudent	2	0.023	93.9	97.2	qwen-coder-parastudent	2	0.015	95.5	96.9

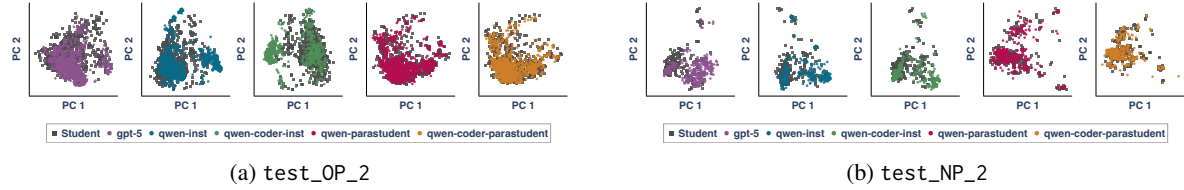


Figure 8: Code embedding visualization on test_OP_2 and test_NP_2. (Experiment 2: with AI tutor feedback). Real student code (gray squares) is compared against gpt-5 (purple circles), qwen-inst (blue circles), qwen-coder-inst (green circles), qwen-parastudent (pink circles), and qwen-coder-parastudent (orange circles).

Table 5: Wasserstein distance between model and student distributions on test_OP_1 and test_NP_1 on Llama-3.1-8B model variants. (Experiment 2: with AI tutor feedback)

(a) test_OP_1						(b) test_NP_1					
Model	Pass Rate ↓	PEP 8 ↓	LOC ↓	AST Depth ↓	AST Width ↓	Model	Pass Rate ↓	PEP 8 ↓	LOC ↓	AST Depth ↓	AST Width ↓
llama-inst	0.138	1.033	0.801	0.511	0.697	llama-inst	0.368	0.404	1.549	0.467	0.743
llama-parastudent	0.025	0.231	0.366	0.304	0.539	llama-parastudent	0.062	0.100	0.546	0.126	0.247

Table 6: Engagement discrimination on the aggregated test_NP split under Experiment 2 (with AI tutor feedback). ROC AUCs measure discrimination between interaction streams with real engagement above versus at or below the 25th- or 50th-percentile threshold. Brackets report 95% confidence intervals. Bold denotes the highest unrounded AUC for each engagement metric and threshold, not statistical significance.

Score	Model	AUC: 25th percentile (95% CI)	AUC: 50th percentile (95% CI)
$\bar{S}_{rel}$	gpt-5	0.74 [0.68, 0.80]	0.77 [0.72, 0.81]
$\bar{S}_{rel}$	qwen-inst	0.75 [0.69, 0.80]	0.76 [0.71, 0.81]
$\bar{S}_{rel}$	qwen-coder-inst	0.75 [0.69, 0.80]	0.77 [0.72, 0.82]
$\bar{S}_{rel}$	qwen-parastudent	0.81 [0.76, 0.86]	0.76 [0.71, 0.81]
$\bar{S}_{rel}$	qwen-coder-parastudent	0.81 [0.75, 0.87]	0.80 [0.75, 0.85]
$\bar{S}_{succ}$	gpt-5	0.51 [0.49, 0.55]	0.52 [0.50, 0.55]
$\bar{S}_{succ}$	qwen-inst	0.57 [0.53, 0.62]	0.55 [0.51, 0.58]
$\bar{S}_{succ}$	qwen-coder-inst	0.58 [0.53, 0.63]	0.55 [0.52, 0.59]
$\bar{S}_{succ}$	qwen-parastudent	0.72 [0.67, 0.78]	0.72 [0.67, 0.77]
$\bar{S}_{succ}$	qwen-coder-parastudent	0.80 [0.75, 0.85]	0.80 [0.75, 0.84]

System message

You are an expert in programming education. These problems come from an introductory programming course where students iteratively revise their code based on feedback. Your task is to analyze **the specified target sentence** in the feedback and explain whether and how it influenced the student's code edits (both additions and deletions), and whether those edits were correct.

You are given:

- The problem statement
- The student's code at time t-1
- The feedback message received at time t-1
- Target sentence to analyze from the feedback
- The student's code at time t
- A summary of the code diff (i.e., the list of lines added and removed between the two versions)

TASK

1. For the specified sentence to analyze, decide whether the student's code edits (added or removed lines) are **grounded in** (i.e., directly motivated by) that sentence.
2. If the sentence is relevant, judge whether the resulting code edit was correct (i.e., whether the change addressed the feedback successfully) by examining the relevant code edits and inferring the student's reasoning or possible misconception.
3. For the target sentence, first provide your rationale, explaining your reasoning clearly before giving boolean decisions.
4. Always output rationales even when the sentence is irrelevant.

OUTPUT FORMAT (JSON)

```
```json
{
 "feedback_analysis": [
 {
 "relevance_rationale": "...",
 "is_relevant": true,
 "success_rationale": "...",
 "is_success": true
 }
]
},
```
```

NOTES

- **'is_relevant'** should be:
 - **'true'** -> the student's code edits were directly motivated by this feedback sentence
 - **'false'** -> the student's code edits were not influenced by this sentence
- **'is_success'** should be:
 - **'true'** -> the edit fixed or improved the targeted issue
 - **'false'** -> the edit was incorrect or introduced new errors
 - **'null'** -> the sentence was irrelevant to the code edit (**'is_relevant = false'**)
- When generating rationales:
 - For **'relevance_rationale'**, focus on **feedback--edit causality**, referring to specific added/removed lines.
 - For **'success_rationale'**, focus on **whether the edit successfully addresses the targeted feedback** and inferred student understanding or misconception.
- Place each rationale immediately before its corresponding boolean field in the JSON.
- Output **JSON only**, with **no commentary** outside the JSON.

User message

PROBLEM INSTRUCTIONS:

{instructions}

FULL FEEDBACK:

{feedback_text}

TARGET SENTENCE TO ANALYZE:

{sentence}

BLOCK TRANSITION:

Block t-1 -> t ({label})

CODE AT TIME t-1:

<code>

{prev_code}

</code>

CODE AT TIME t:

<code>

{next_code}

</code>

CODE DIFF SUMMARY:

ADDED:

{added_line}

REMOVED:

{removed_line}

Figure 9: Prompt used for the LLM judge to attribute student code edits to individual feedback sentences.

Table 7: **Pairwise agreement between the LLM judge and two human annotators (A1 and A2), and between annotators, for *Relevant* and *Success* labels on 50 randomly sampled feedback–code edit pairs. Positive and negative agreement are computed with respect to the human annotator for LLM–human comparisons and A1 for the A1–A2 comparison.**

| Comparison Pair | Relevant | | | | Success | | | |
|-----------------|----------|----------|------|----------|----------|----------|------|----------|
| | Positive | Negative | All | κ | Positive | Negative | All | κ |
| LLM–A1 | 0.96 | 0.64 | 0.80 | 0.60 | 0.68 | 0.97 | 0.86 | 0.69 |
| LLM–A2 | 0.93 | 0.71 | 0.84 | 0.66 | 0.60 | 0.93 | 0.80 | 0.56 |
| A1–A2 | 0.96 | 0.80 | 0.88 | 0.76 | 0.89 | 0.90 | 0.90 | 0.79 |