

CAP: Cross-Ability Preferences to Align Student Simulators for Item Difficulty Prediction

Nigel Fernandez¹, Alexander Scarlatos¹, Christopher Ormerod², Andrew Lan¹

¹University of Massachusetts Amherst, ²College Board

{nigel,ajscarlatos,andrewlan}@cs.umass.edu, christopher.ormerod@gmail.com

Abstract

Predicting the difficulty of open-ended assessment items from only a handful of labeled examples is a central challenge in educational measurement. Simulation-based methods address this low-resource setting by generating a population of synthetic student responses, scoring them, and fitting an item response theory (IRT) model to recover the item's difficulty. Their accuracy, however, hinges on a simulator whose response quality faithfully tracks the prompted student ability, a relationship that supervised finetuning (SFT) captures only weakly. We present CAP (Cross-Ability Preferences), which aligns a large language model (LLM) student simulator with direct preference optimization (DPO) over preference pairs formed directly from students' IRT ability gaps: conditioned on a student's ability, that student's real response is preferred over a response by a student of a sufficiently different ability on the same item. On the CodeWorkout programming dataset, CAP surpasses all direct difficulty-prediction baselines, and at the 1B scale it outperforms the state-of-the-art simulator SMART on every difficulty metric while generating more distributionally realistic responses; at 3B it performs comparably.

1 Introduction

Estimating the difficulty parameters of assessment items is one of the most difficult problems in computational educational measurement (AlKhazaey et al., 2024). Item difficulty, which captures how challenging a question is for a population of students, is a core parameter of item response theory (IRT) (Rasch, 1960; Lord, 1980) and underpins many downstream applications, including adaptive testing, personalized item recommendation, and curriculum design. Traditionally, an item's difficulty is estimated using field testing, which involves exposing the item to many real students and fitting an IRT model to their responses. Apart from

being costly and time-consuming, this approach presents a security challenge in high-stakes settings and, more fundamentally, cannot be applied to *new* items in the cold-start setting, where an item must be calibrated *before* any student has answered it.

A growing body of work predicts item difficulty directly from item text, using large language models (LLMs) and handcrafted linguistic features (AlKhazaey et al., 2024; Benedetto et al., 2021; Gombert et al., 2024). These methods work well for multiple-choice questions when a large bank of previously calibrated items is available to learn from. However, their only training signal is the observed item difficulties, so they require many labeled items and degrade in the *low-resource* regime of only a handful of items.

An appealing alternative is to *simulate* student responses. Rather than predicting difficulty from item text alone, these methods generate a population of synthetic responses to the item, score them, and fit an IRT model to the resulting scores to recover the item's difficulty (Lalor et al., 2019; He-Yueya et al., 2024; Scarlatos et al., 2025). Because they couple the item's content with simulated response *behavior*, such methods remain effective even when very few labeled items are available, which makes them well suited to the low-resource setting. Their central challenge is to make the simulated responses faithfully reflect a prompted student ability: the difficulty estimate is only as good as the simulator's capacity to mimic how students of different ability would actually answer.

This challenge is especially acute for *open-ended* items, such as programming problems, where a response is free-form (e.g., a code submission) and graded for partial correctness, and where even students of similar ability produce highly diverse answers. A simulator finetuned only to imitate real responses conditions weakly on the prompted ability, since its objective rewards reproducing response *text* rather than tying response *quality* to ability.

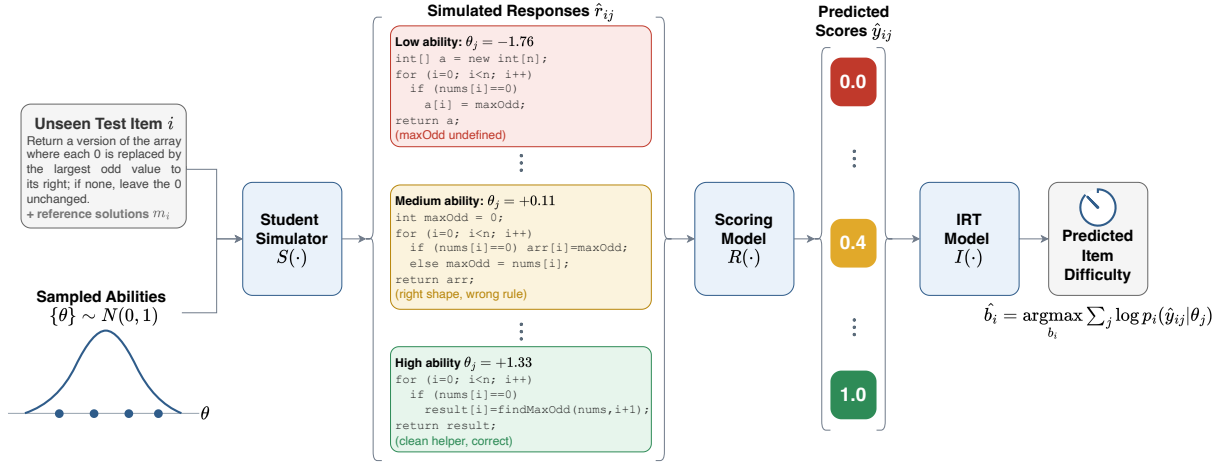


Figure 1: The CAP pipeline: an ability-conditioned LLM simulator $S(\cdot)$ generates student responses to an unseen item, a scoring model $R(\cdot)$ scores them, and an IRT model $I(\cdot)$ is refit on the scores to estimate the item’s difficulty.

The central question we address is how to make the simulator’s response quality follow the prompted student ability without costly additional machinery. We provide an expanded discussion of related work in Appendix A.

1.1 Contributions

In this paper, we present **CAP**¹ (Cross-Ability Preferences), a method for aligning an LLM-based student simulator so that the quality of its generated responses tracks the prompted student ability. A simulator trained only by supervised finetuning (SFT) conditions weakly on ability; CAP instead aligns it with direct preference optimization over pairs that contrast the real responses of students of *different* abilities, selected directly by their IRT ability gap. This explicit, ability-contrastive signal is precisely what SFT lacks. Our main contributions are as follows:

1. We form **cross-ability preference pairs** *directly* from students’ IRT abilities (pairing each response against a response by a student of a sufficiently different ability), and show that this pairing, trained with standard DPO and no additional loss machinery, improves a simulated-student difficulty predictor.
2. We conduct extensive experiments on the public CodeWorkout programming dataset at two model scales (1B and 3B), against a variety of direct and simulation-based baselines.
3. CAP outperforms all direct difficulty-prediction baselines. At the 1B scale it

outperforms the state-of-the-art SMART on every difficulty-prediction metric and generates more distributionally realistic responses; at 3B it performs comparably.

2 Background: Item Response Theory

Item response theory (IRT) (Rasch, 1960; Lord, 1980) is a framework for jointly estimating scalar-valued student abilities and item parameters from a set of scored student responses to items. Each student j is represented by a scalar ability $\theta_j \in \mathbb{R}$ and each item i by a scalar difficulty $b_i \in \mathbb{R}$, where a correct response becomes more likely as a student’s ability grows relative to an item’s difficulty. The simplest model, the one-parameter logistic (1PL) or Rasch model, gives the probability of a correct (binary) response as

$$P_i^{\text{1PL}}(y_{ij} | \theta_j) = \sigma(\theta_j - b_i), \quad (1)$$

where $\sigma(\cdot)$ is the logistic function.

The student responses we work with are open-ended and graded on a continuous scale, i.e., $y_{ij} \in [0, 1]$, so that a response may be partially correct rather than strictly right or wrong. Following Scarlatos et al. (2025), we model each response score with a continuous Bernoulli distribution (Loaiza-Ganem and Cunningham, 2019), which extends the 1PL model to continuous-valued outcomes:

$$P_i^{\text{Bern.}}(y_{ij} | \theta_j) = P_i^{\text{1PL}}(Y=1 | \theta_j)^{y_{ij}} P_i^{\text{1PL}}(Y=0 | \theta_j)^{1-y_{ij}}. \quad (2)$$

3 Problem Formulation

We assume access to a dataset of human-scored student responses to open-ended items. Each item i

¹Code: <https://github.com/umass-ml4ed/cap>

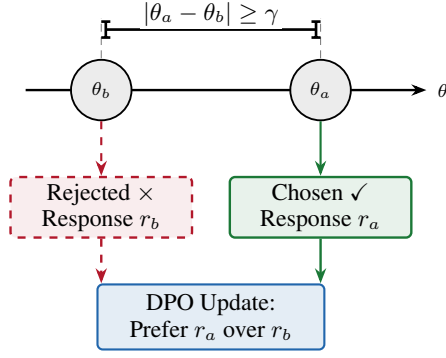


Figure 2: CAP’s cross-ability preference pair, formed offline from *real* student responses: the chosen r_a by a student of ability θ_a is preferred over the rejected r_b by a student of a *sufficiently different* ability (more or less able), $|\theta_a - \theta_b| \geq \gamma$. During DPO, both responses are conditioned on the same prompt (specifying ability θ_a).

is associated with question text q_i , optional meta-data m_i (e.g., reference solutions), and a set of human-scored responses $D_i = \{(r_{ij}, y_{ij})\}$, where y_{ij} is the score that student j received for response r_{ij} . Fitting an IRT model to all responses yields ground-truth student abilities θ_j and item difficulties b_i . Our goal is to predict the difficulty b_i of a previously unseen item, before any student has responded to it.

4 Methodology

Our method, CAP, predicts the difficulty of an unseen item without collecting any real student responses to it. Following a setup similar to SMART (Scarlatos et al., 2025) and as illustrated in Figure 1, it proceeds in three stages, each parameterized by a learned model: a *student simulator* $S(\cdot)$, a generative LLM that produces a population of student responses to the item, each response conditioned on its own sampled student ability; a *scoring model* $R(\cdot)$, a generative LLM that scores the simulated responses; and an *IRT model* $I(\cdot)$, which we refit on the resulting scores to estimate the item’s difficulty. The crux of CAP is how we train the simulator $S(\cdot)$ so that the quality of the responses it generates faithfully follows the prompted ability, which we detail in Section 4.2.

4.1 Student Simulator

We parameterize the student simulator $S(\cdot)$ with a generative LLM and finetune it to generate a student’s response given the item and a target ability. For item i and a student of ability θ_j , we construct an input prompt from the question text q_i , the meta-

data m_i , and the ability θ_j (encoded as its numeric value rounded to four decimals); we then train the model by supervised finetuning (SFT) to maximize the likelihood of the ground-truth response r_{ij} . The exact prompt is given in Appendix E.

Although θ_j appears in the prompt, SFT ties response quality to ability only *implicitly* and *weakly*, for three reasons: (i) its objective maximizes the likelihood of the response *text*, with no term forcing response quality to increase with θ_j (precisely what the θ -Align metric of Section 5.2 measures); (ii) the high variance of real responses at any single ability makes adjacent-ability distributions overlap, so only well-separated abilities are reliably distinguishable; and (iii) θ_j is a lone scalar in a long prompt, so the loss can be driven down by modeling ability-agnostic code structure while barely attending to it. The SFT simulator therefore follows the prompted ability poorly.

4.2 CAP: Cross-Ability Preferences

To supply the ability signal SFT lacks (above), we align the simulator with direct preference optimization (DPO) (Rafailov et al., 2023), continuing training from the SFT simulator and keeping it frozen as the reference policy. What sets CAP apart is how it forms the preference pairs; the rest is a standard DPO recipe. The guiding intuition is simple: when prompted at a given ability, the simulator should answer the item as a real student of that ability would, and unlike a student of a very different ability. We form preference pairs offline from *real* student responses: for a response r_a written by a student of ability θ_a , we take r_a as the preferred (chosen) response and pair it with a dispreferred (rejected) response r_b , a real response by a student of a *sufficiently different* ability (more or less able), $|\theta_a - \theta_b| \geq \gamma$ (Figure 2). Preferring the response that matches the prompted ability over one from a far-different ability is what drives the simulator’s response quality to follow that ability. Concretely, for a chosen response r_a the set of admissible rejected responses to the same item is

$$\mathcal{R}(r_a) = \{r_b : |\theta_a - \theta_b| \geq \gamma\}, \quad (3)$$

from which we sample up to m to form preference pairs with r_a . The cross-ability contrast is thus formed *directly* from the student-ability gap, requiring only real student responses and their IRT abilities, with no response scoring, likelihood computation, or other auxiliary machinery. SMART (Scarlatos et al., 2025) instead forms its pairs from an

IRT-likelihood gap on a response’s score; since that likelihood is only a score-mediated proxy for ability, it contrasts ability *indirectly*, whereas CAP does so directly. We train on these pairs with the standard DPO objective (Rafailov et al., 2023), conditioning both the chosen and rejected response on the same prompt (specifying ability θ_a).

4.3 Scoring Model

To estimate item difficulty from simulated responses we must score them, including responses to items not seen during training. We parameterize the scoring model $R(\cdot)$ with a generative LLM and finetune it to predict a response’s score from the question q_i , the metadata m_i , and the response r_{ij} (prompt in Appendix E). Because CodeWorkout scores are continuous, we read a score by interpolating between the logits of the “1” and “0” tokens at the answer position, following SMART (Scarlatos et al., 2025),

$$\hat{y}_{ij} = \sigma(R(1 \mid x_{ij}) - R(0 \mid x_{ij})), \quad (4)$$

where x_{ij} is the scoring prompt and $R(\cdot \mid x_{ij})$ its output logits. We train with binary cross-entropy against the ground-truth scores.

4.4 Simulation-Based Difficulty Prediction

As illustrated in Figure 1, with the trained simulator and scorer in hand we predict an unseen item’s difficulty as follows. We first draw a population of n simulated students whose abilities match the training-set ability distribution. For each drawn ability, we prompt the simulator $S(\cdot)$ to generate a response to the test item using nucleus sampling, and score it with $R(\cdot)$. We then refit the IRT model $I(\cdot)$ on these scored simulated responses and take the resulting estimate $\hat{b}_i$ as our predicted difficulty for the item. We defer the details of ability sampling and IRT refitting to Appendix B.

5 Experimental Evaluation

5.1 Datasets

We evaluate on **CodeWorkout** (DataShop, 2021), a popular and publicly available dataset of student submissions to introductory Java programming problems, originally released for the 2nd CSEDM Data Challenge (Organizers, 2021). This dataset, used for student modeling in open-ended programming tasks (Duan et al., 2026a, 2025; Fernandez and Lan, 2024), comprises 50 problems; following common practice (Duan et al., 2026b),

we keep only each student’s first attempt at each problem, yielding 10,834 responses from 246 students. Each response received a continuous score $y_{ij} \in [0, 1]$ equal to the fraction of unit tests it passed. As item metadata m_i , we include two reference solutions per problem, generated by the LLM-based method of Duan et al. (2026b), following SMART (Scarlatos et al., 2025), which likewise provides such reference solutions as additional context to the simulator and scorer. We fit a continuous-Bernoulli IRT model to all responses to obtain ground-truth abilities θ_j and difficulties b_i , and split the items into difficulty-stratified 5-fold cross-validation folds (60/20/20 train/validation/test). To prevent leakage from unseen items, we fit three separate IRT models, on the train, train-plus-validation, and full data, used respectively to obtain the abilities that condition the simulator, as the validation-time ground truth, and as the test-time ground truth. We provide further details on the data and cross-validation splits in Appendix C and on IRT fitting in Appendix B.4.

5.2 Metrics

We assess two aspects of each method. For **item difficulty prediction**, following Scarlatos et al. (2025), we report the Pearson correlation coefficient (PCC) and Spearman rank correlation coefficient (SCC) between predicted and ground-truth item difficulties, together with the root-mean-square error (RMSE). For **response simulation quality**, we measure how faithful the generated responses are to real student ones. Following Scarlatos et al. (2025), we report θ -Align, the Spearman correlation between a simulated student’s input ability and the predicted score of its generated response, which captures how well the simulator follows the prompted ability. We also compare the generated and real response distributions in a sentence-embedding space (SBERT (Reimers and Gurevych, 2019)) using MAUVE (Pillutla et al., 2021), FID (Heusel et al., 2017), and Div. KL (Bui et al., 2025), the KL divergence between histograms of pairwise response similarity.

5.3 Baselines

Following prior work (Scarlatos et al., 2025), we compare CAP against two families of baselines; exact hyperparameters and implementation details for all of them are in Appendix B.

Method	PCC $\uparrow$	SCC $\uparrow$	RMSE $\downarrow$
Direct Item Difficulty Prediction			
Diff SFT (ModernBERT)	0.295 $\pm$ 0.14	0.353 $\pm$ 0.18	0.660 $\pm$ 0.06
Random Forest (SBERT)	0.313 $\pm$ 0.38	0.350 $\pm$ 0.46	0.633 $\pm$ 0.18
kNN Mean (SBERT, $k=5$)	0.225 $\pm$ 0.32	0.238 $\pm$ 0.40	0.683 $\pm$ 0.13
ICL Random (Llama-3.1-70B)	-0.025 $\pm$ 0.30	-0.051 $\pm$ 0.33	0.788 $\pm$ 0.12
ICL kNN (Llama-3.1-70B, $k=2$)	0.193 $\pm$ 0.28	0.153 $\pm$ 0.33	0.700 $\pm$ 0.11
Student Simulation (Llama-3.2-1B)			
Response Gen ZS	0.269 $\pm$ 0.34	0.358 $\pm$ 0.23	0.659 $\pm$ 0.15
Response Gen SFT	0.364 $\pm$ 0.12	0.307 $\pm$ 0.18	0.627 $\pm$ 0.06
SMART	0.404 $\pm$ 0.26	0.375 $\pm$ 0.26	0.595 $\pm$ 0.14
CAP (ours)	0.428 $\pm$ 0.19	0.469 $\pm$ 0.18	0.590 $\pm$ 0.10
Student Simulation (Llama-3.2-3B)			
Response Gen ZS	0.290 $\pm$ 0.22	0.292 $\pm$ 0.25	0.658 $\pm$ 0.10
Response Gen SFT	0.392 $\pm$ 0.22	0.273 $\pm$ 0.13	0.607 $\pm$ 0.12
SMART	0.516 $\pm$ 0.34	0.532 $\pm$ 0.29	0.528 $\pm$ 0.18
CAP (ours)	0.493 $\pm$ 0.18	0.508 $\pm$ 0.09	0.559 $\pm$ 0.09

Table 1: Item difficulty prediction on CodeWorkout (5-fold CV, mean $\pm$ std across folds). Higher PCC/SCC and lower RMSE are better; among the simulation methods, **best** and second best per scale block are marked. CAP is best among 1B methods on all three metrics; at 3B SMART is best.

Direct difficulty prediction. These methods predict difficulty from item text alone. **Diff SFT** finetunes ModernBERT-base (Warner et al., 2024) to regress item difficulty. **Random Forest** and **kNN Mean** ($k=5$) operate on SBERT (Reimers and Gurevych, 2019) embeddings of the item, the former fitting a regressor and the latter averaging the difficulties of the nearest training items. **ICL Random** and **ICL kNN** prompt Llama-3.1-70B-Instruct (Grattafiori et al., 2024) with $k=2$ in-context examples (selected at random or by SBERT similarity, respectively) and chain-of-thought reasoning before predicting a difficulty.

Simulation-based. These methods follow the same simulate-score-fit pipeline as CAP. **Response Gen ZS** prompts the base LLM zero-shot, mapping abilities to ten descriptive buckets. **Response Gen SFT** is the SFT simulator of Section 4 without preference alignment. **SMART** (Scarlatos et al., 2025) adds DPO over pairs selected by the IRT likelihood of a response’s score under the prompted ability.

5.4 Experimental Setup

We use Llama-3.2-Instruct (Grattafiori et al., 2024) as the base model for both the simulator and the scorer, loaded in bfloat16 and finetuned with LoRA (Hu et al., 2022), and run all experiments at both 1B and 3B parameters to study how the methods scale. For CAP we set the ability-gap threshold γ to 1.5 (1B) and 3.0 (3B); the number of rejected responses per chosen response, m (1 at 1B, 5 at 3B), sets the pair count, which we cap for fairness so it does not substantially exceed SMART’s per-

fold budget. All simulation-based methods (Response Gen SFT, SMART, and CAP) share the *same* SFT simulator and scoring model, and CAP uses the same standard DPO loss as SMART. Because the original SMART code was not released, our SMART and all other baselines are independent reproductions from its published description; on the difficulty-prediction metrics, the task we ultimately care about, our reproduction matches or exceeds the originally reported SMART numbers at both model sizes, giving us confidence that the comparison is faithful. We report all results as the mean and standard deviation over the five cross-validation folds; full hyperparameters and inference settings are in Appendix B.

6 Results, Analysis, and Discussion

6.1 Quantitative Results

CAP outperforms direct baselines. As Table 1 shows, the direct text-based methods are weak on programming difficulty: the best of them (Random Forest, Diff SFT) reaches a PCC of only ≈ 0.31 with very large variance, while ICL Random hovers near zero. CAP comfortably surpasses them on every metric at both model sizes.

Student simulation is essential in the low-resource setting. With only ~ 50 items, the direct methods have too little signal to learn from, since their supervision is the item difficulties alone. The simulation pipeline instead recovers difficulty from the *content* of generated responses, coupling item text with response behavior, which is what lets it

Method	MAUVE $\uparrow$	FID $\downarrow$	Div. KL $\downarrow$	θ -Align $\uparrow$
Student Simulation (Llama-3.2-1B)				
Response Gen ZS	0.004 $\pm$ 0.00	0.344 $\pm$ 0.02	0.929 $\pm$ 0.35	0.021 $\pm$ 0.01
Response Gen SFT	0.188 $\pm$ 0.07	0.028 $\pm$ 0.01	0.029 $\pm$ 0.02	0.225 $\pm$ 0.03
SMART	0.134 $\pm$ 0.08	0.050 $\pm$ 0.03	0.071 $\pm$ 0.09	0.581 $\pm$ 0.13
CAP (ours)	0.168 $\pm$ 0.05	0.035 $\pm$ 0.01	0.036 $\pm$ 0.02	0.490 $\pm$ 0.13
Student Simulation (Llama-3.2-3B)				
Response Gen ZS	0.004 $\pm$ 0.00	0.381 $\pm$ 0.02	0.794 $\pm$ 0.37	-0.074 $\pm$ 0.02
Response Gen SFT	0.216 $\pm$ 0.07	0.023 $\pm$ 0.01	0.030 $\pm$ 0.01	0.296 $\pm$ 0.03
SMART	0.122 $\pm$ 0.03	0.038 $\pm$ 0.01	0.027 $\pm$ 0.01	0.668 $\pm$ 0.10
CAP (ours)	0.108 $\pm$ 0.02	0.046 $\pm$ 0.01	0.043 $\pm$ 0.01	0.681 $\pm$ 0.03

Table 2: Simulation quality on CodeWorkout (5-fold CV, mean $\pm$ std across folds). **Best** and second best per scale block are marked. Unaligned SFT is most realistic but conditions poorly on ability (low θ -Align); among aligned simulators, CAP improves realism over SMART at 1B while trading θ -Align, and attains the highest θ -Align at 3B.

generalize from so few items.

CAP outperforms SMART at 1B and is comparable at 3B. At 1B, CAP is the best simulation method on all three difficulty metrics (PCC 0.428 vs. SMART’s 0.404, SCC 0.469 vs. 0.375, and a lower RMSE). At 3B the two perform comparably: CAP’s PCC and SCC means are marginally lower (0.493 vs. 0.516 PCC). More broadly, cross-ability preferences help most at the smaller model scale and perform on par with the state of the art at the larger scale. This 1B advantage is specific to the few-pair regime: CAP uses a single rejected response per chosen response ($m=1$), and raising m until the pair count matches SMART’s lowers CAP below SMART at 1B. We attribute this to a regularization effect: the few, gentle cross-ability contrasts at $m=1$ suit the smaller model, whereas the 3B model benefits from more pairs ($m=5$). We emphasize that, with only ~ 10 test items per fold, the per-fold standard deviation is large; none of these differences is statistically significant, and we report means for comparison.

Cross-ability pairing improves distributional realism while trading ability alignment. Table 2 compares the simulators’ response distributions. Among the aligned simulators, CAP improves over SMART at 1B on all three distributional metrics (MAUVE, FID, and Div. KL), but at a lower θ -Align (0.490 vs. 0.581): its responses are more realistic in aggregate yet slightly less monotonically tied to the prompted ability. At 3B the picture reverses on θ -Align, where CAP is highest (0.681 vs. 0.668), though SMART is more realistic distributionally. The unaligned SFT simulator is the most realistic of all but conditions poorly on ability, confirming that distributional realism and ability alignment are distinct axes.

6.2 Qualitative Evaluation

To complement the quantitative results, we inspect the simulated responses and find that CAP’s generations exhibit the features one would expect of a student at the prompted ability, mirroring real student submissions. At low ability, CAP tends to produce incomplete or buggy code (missing return statements, undeclared variables, or incorrect logic) that earns little credit; at medium ability, the code has the right overall structure but contains a subtle bug and earns partial credit; and at high ability, the code is clean and correct, often factoring out a helper function. As a result, the predicted score of CAP’s generations rises with the prompted ability, the qualitative counterpart of its high θ -Align. The SFT simulator, by contrast, generates code whose quality is largely decoupled from the conditioned ability, sometimes producing a fully correct solution at low ability and a broken one at high ability. We provide a side-by-side example on a representative CodeWorkout item in Appendix D.

7 Conclusions and Future Work

We presented CAP, which aligns an LLM-based student simulator for item difficulty prediction by forming DPO preference pairs that contrast the responses of students of *different* abilities, selected directly by their ability gap. On CodeWorkout, this pairing outperforms all direct baselines and improves over the state-of-the-art SMART at 1B, while performing comparably at 3B. We leave extending the approach to other domains and larger student simulators, and exploring online preference pair generation, for future work.

Limitations

Our work has several limitations. First, we evaluate on a single public dataset (CodeWorkout) with only ~ 50 items, which yields large per-fold variance (~ 10 test items per fold); consequently, our gains over SMART are not statistically significant, and the comparison rests on mean performance. Second, we restrict the student simulator to 1B and 3B models, and whether the approach scales to larger simulators is untested. Third, response simulation quality remains an open challenge: the unaligned SFT simulator produces the most distributionally realistic responses, leading both CAP and SMART on MAUVE and FID at both scales, so the realism of aligned simulators has substantial room to improve.

References

- Samah AlKhuyaey, Floriana Grasso, Terry R Payne, and Valentina Tamma. 2024. Text-based question difficulty prediction: A systematic review of automatic approaches. *International Journal of Artificial Intelligence in Education*, 34(3):862–914.
- Luca Benedetto, Giovanni Aradelli, Paolo Cremonesi, Andrea Cappelli, Andrea Giussani, and Roberto Turin. 2021. On the application of transformers for estimating the difficulty of multiple-choice questions from text. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 147–157, Online. Association for Computational Linguistics.
- Ngoc Bui, Hieu Trung Nguyen, Shantanu Kumar, Julian Theodore, Weikang Qiu, Viet Anh Nguyen, and Rex Ying. 2025. Mixture-of-personas language models for population simulation. *Preprint*, arXiv:2504.05019.
- Matthew Byrd and Shashank Srivastava. 2022. Predicting difficulty and discrimination of natural language questions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 119–130, Dublin, Ireland. Association for Computational Linguistics.
- DataShop. 2021. Dataset: Codeworkout data spring 2019. Online: <https://pslcdatashop.web.cmu.edu/Files?datasetId=3458>.
- 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*, pages 238–248.
- Zhangqi Duan, Nigel Fernandez, and Andrew S. Lan. 2026a. KASER: Knowledge-aligned student error simulator for open-ended coding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*.
- Zhangqi Duan, Nigel Fernandez, A. Narayanan, Mohammad Hassany, R. Alencar, Peter Brusilovsky, Bita Akram, and Andrew S. Lan. 2026b. Automated knowledge component generation and knowledge tracing for coding problems. In *Findings of the Association for Computational Linguistics: ACL 2026*.
- Wanyong Feng, Peter Tran, Stephen Sireci, and Andrew S. Lan. 2025. Reasoning and sampling-augmented mcq difficulty prediction via llms. *Preprint*, arXiv:2503.08551.
- Nigel Fernandez and Andrew S. Lan. 2024. Interpreting latent student knowledge representations in programming assignments. In *Proceedings of the 17th International Conference on Educational Data Mining*, pages 933–940, Atlanta, Georgia, USA. International Educational Data Mining Society.
- Sebastian Gombert, Lukas Menzel, Daniele Di Mitri, and Hendrik Drachsler. 2024. Predicting item difficulty and item response time with scalar-mixed transformer encoder models and rational network regression heads. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 483–492, Mexico City, Mexico. Association for Computational Linguistics.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Joy He-Yueya, Wanjing Anya Ma, Kanishk Gandhi, Benjamin W Domingue, Emma Brunskill, and Noah D Goodman. 2024. Psychometric alignment: Capturing human knowledge distributions via language models. *arXiv preprint arXiv:2407.15645*.
- Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.
- 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*.
- Zhenya Huang, Qi Liu, Enhong Chen, Hongke Zhao, Mingyong Gao, Si Wei, Yu Su, and Guoping Hu. 2017. Question difficulty prediction for reading problems in standard tests. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI’17, page 1352–1359. AAAI Press.
- John P. Lalor, Hao Wu, and Hong Yu. 2019. Learning latent parameters without human response patterns: Item response theory with artificial crowds. In

- Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4249–4259, Hong Kong, China. Association for Computational Linguistics.
- Gabriel Loaiza-Ganem and John P Cunningham. 2019. The continuous bernoulli: fixing a pervasive error in variational autoencoders. *Advances in Neural Information Processing Systems*, 32.
- Frederic M. Lord. 1980. *Applications of Item Response Theory to Practical Testing Problems*. Lawrence Erlbaum Associates.
- Challenge Organizers. 2021. The 2nd cseddm data challenge. Online: <https://sites.google.com/ncsu.edu/csedm-dc-2021/>.
- Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Advances in Neural Information Processing Systems*, 34:4816–4828.
- 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*. In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Georg Rasch. 1960. Probabilistic models for some intelligence and attainment tests. *Copenhagen: Institute of Education Research*.
- Nils Reimers and Iryna Gurevych. 2019. *Sentence-bert: Sentence embeddings using siamese bert-networks*. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Alexander Scarlatos, Nigel Fernandez, Christopher Ormerod, Susan Lottridge, and Andrew S. Lan. 2025. SMART: Simulated students aligned with item response theory for question difficulty prediction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25082–25105.
- Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. *Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference*. Preprint, arXiv:2412.13663.
- Victoria Yaneva, Kai North, Peter Baldwin, Le An Ha, Saed Rezayi, Yiyun Zhou, Sagnik Ray Choudhury, Polina Harik, and Brian Clauser. 2024. *Findings from the first shared task on automated prediction of difficulty and response time for multiple-choice questions*. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 470–482, Mexico City, Mexico. Association for Computational Linguistics.
- Eric Zelikman, Wanjing Ma, Jasmine Tran, Diyi Yang, Jason Yeatman, and Nick Haber. 2023. *Generating and evaluating tests for k-12 students with language model simulations: A case study on sentence reading efficiency*. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2190–2205, Singapore. Association for Computational Linguistics.

A Related Work

A.1 Direct Item Difficulty Prediction Methods

Item difficulty prediction has been studied extensively (AlKhuyaey et al., 2024), most often for multiple-choice questions (Yaneva et al., 2024; Feng et al., 2025). Approaches range from regression on handcrafted linguistic features (Byrd and Srivastava, 2022) to finetuned pretrained encoders such as BERT (Benedetto et al., 2021; Gombert et al., 2024) and specialized neural architectures (Huang et al., 2017). All predict difficulty from item text alone, supervised entirely by observed item difficulties, so they are most accurate with a large bank of labeled items. In the low-resource regime we target (only tens of items), this signal is too sparse, and our experiments (Section 6) confirm that every such baseline is substantially outperformed by our method, CAP, on programming items.

A.2 Simulation-Based Item Difficulty Prediction Methods

A growing body of work predicts item difficulty by simulating student responses and fitting an IRT model to the simulated outcomes (Lalor et al., 2019; He-Yueya et al., 2024; Scarlatos et al., 2025). These methods overgenerate responses, using ensembles of language models, simulators finetuned on students’ prior responses, or persona- and psychometric-style prompting (Zelikman et al., 2023; He-Yueya et al., 2024), and recover difficulty from the aggregated scores. Because they couple item content with response behavior, they remain effective with few labeled items, which is why we adopt this paradigm. In the same open-ended coding setting, KASER (Duan et al., 2026a) also aligns an LLM to simulate student responses, but targets reproducing student errors rather than predicting item difficulty.

Closest to our work is SMART (Scarlatos et al., 2025), which we build upon. SMART aligns its simulator with DPO over preference pairs selected by the IRT likelihood of a response’s score under the prompted ability; since score correlates with ability, it contrasts abilities only *indirectly*, mediated by the score. CAP instead forms cross-ability pairs *directly* from the student-ability gap, requiring no response scoring or IRT-likelihood computation, and otherwise trains with the same standard DPO loss, making its alignment signal both simpler to construct and more directly tied to ability.

B Experimental Details

B.1 Student Simulation

We parameterize both the student simulator and the scoring model with Llama-3.2-Instruct (1B and 3B), loaded in bfloat16 and finetuned with LoRA (rank 128, $\alpha=64$, dropout 0.05) applied to all attention and MLP projections. We finetune the simulator for 3 epochs with AdamW at a learning rate of 5×10^{-5} , linear warmup over the first 10% of steps, weight decay 10^{-2} , gradient clipping at 1.0, an effective batch size of 64 (micro-batch 8 with gradient accumulation), and a maximum sequence length of 1024; the prompted ability θ_j is encoded as its value rounded to four decimals. We finetune the scoring model with the same LoRA configuration for 5 epochs (AdamW, learning rate 5×10^{-5} , effective batch size 64), minimizing binary cross-entropy between the interpolated score (Eq. 4) and the ground-truth score.

We then align the simulator with DPO for 1 epoch using $\beta=0.1$, a learning rate of 10^{-5} , and an effective batch size of 64, keeping the SFT model frozen as the reference policy. CAP forms cross-ability pairs with an ability-gap threshold γ of 1.5 at 1B and 3.0 at 3B, and samples m rejected responses per chosen response ($m=1$ at 1B, $m=5$ at 3B); the rejected responses are drawn at random among far-ability responses, and the resulting pair count is capped so that it does not substantially exceed SMART’s per-fold budget. The DPO loss itself is the standard one, identical to SMART’s; the methods differ only in which responses are paired: SMART instead selects negatives using an IRT-likelihood threshold tuned on validation per scale ($\epsilon=0.2$ at 1B and 0.5 at 3B, with $m=3$ negatives per positive).

For the 3B models we reduce the per-device batch sizes through gradient accumulation (micro-

batch 4 for the simulator and scorer, 2 for DPO, and a generation batch of 32) to fit on a single NVIDIA L40S 48 GB GPU, while preserving the effective batch size.

At inference we draw $n=1000$ simulated students per item (their abilities sampled from a 50-bucket histogram of the training-set abilities) and generate one response per student with nucleus sampling ($T=0.7$, $p=0.95$), decoding up to 500 new tokens.

B.2 Baselines

Diff SFT finetunes ModernBERT-base with a regression head, minimizing mean-squared error against item difficulty, for 40 epochs with AdamW at a learning rate of 2×10^{-5} and a maximum sequence length of 1024. **Random Forest** and **kNN Mean** operate on SBERT embeddings of the item (its question text and reference solutions): the former fits a scikit-learn random-forest regressor with max_features set to the training-set size and otherwise default settings, and the latter predicts the mean ground-truth difficulty of the $k=5$ nearest training items by cosine similarity. **ICL Random** and **ICL kNN** prompt Llama-3.1-70B-Instruct (loaded in 4-bit NF4) with $k=2$ in-context examples (chosen at random or by SBERT similarity, respectively) and ask it to reason step by step before emitting a line of the form “Difficulty: {number}”, which we parse. **Response Gen ZS** uses the untrained base LLM, mapping each ability to one of ten descriptive buckets ranging from “Minimal” ($\theta < -2.5$) to “Mastery” ($\theta \geq 2.5$), and otherwise reuses the same scoring model and IRT pipeline as the trained simulators.

We use the multi-qa-mpnet-base-dot-v1 SBERT model for all embeddings (random forest, kNN, ICL-kNN retrieval, and the simulation metrics). The simulation-quality metrics are computed as follows: MAUVE using its official implementation; FID as the Fréchet distance between Gaussians fit to the unit-normalized SBERT embeddings of the generated and real responses; Div. KL as the KL divergence between 50-bin histograms of the pairwise cosine similarities within each set; and θ -Align as the Spearman correlation between each simulated student’s input ability and the predicted score of its generated response.

B.3 Difficulty Normalization

Following prior work (Scarlatos et al., 2025), for all methods (including the baselines) we transform the

predicted item difficulties to have the same mean and standard deviation as the training-set ground-truth difficulties; this improves the error metrics when predictions are correlated with but shifted or scaled relative to the ground truth. Concretely, each prediction is rescaled as

$$\hat{b}_i \leftarrow \frac{\sigma_{\text{GT}}}{\sigma_{\text{pred}}} (\hat{b}_i - \mu_{\text{pred}}) + \mu_{\text{GT}}, \quad (5)$$

where μ_{pred} and σ_{pred} are the mean and standard deviation of the predicted difficulties and $\mu_{\text{GT}}, \sigma_{\text{GT}}$ those of the training-set ground-truth difficulties. We apply this transformation identically to every method, so it does not affect the comparison between them.

B.4 IRT Model

We model item responses with a continuous-Bernoulli 1PL (Rasch) model; for identifiability we impose a location constraint, shifting the parameters so that the mean student ability is zero. We fit the ground-truth abilities and difficulties by maximum likelihood over all responses, using AdamW with a learning rate of 10^{-3} , no weight decay, a batch size of 256, for 200 epochs; we omit the continuous-Bernoulli normalizing constant and optimize the unnormalized log-likelihood, equivalently a soft-label binary cross-entropy. At inference, to estimate the difficulty of an unseen item, we hold the (rounded) simulated-student abilities fixed and recover the single item difficulty b_i by a one-dimensional Newton’s method; the objective is convex in b_i , so this converges reliably. Following SMART (Scarlatos et al., 2025), we round each simulated student’s ability to one decimal place when linking responses; this keeps the number of distinct ability parameters small, so that each ability value is shared by many simulated students when we refit the item difficulty.

Following prior work (Scarlatos et al., 2025), to prevent leakage from the test items we fit three separate IRT models per fold: one on the training items only (used to obtain the abilities that condition the simulator), one on the training plus validation items (used as the validation-time ground truth), and one on all items (used as the test-time ground truth), so that the validation and test targets never influence each other.

C Dataset Details

CodeWorkout is a publicly available dataset of student submissions to 50 introductory Java pro-

gramming problems, released for the 2nd CSEDMD Data Challenge. Students may submit multiple times after receiving autograder feedback; following common practice we keep only each student’s *first* submission per problem, which yields 10,834 responses from 246 students. Each submission receives a continuous score in $[0, 1]$ equal to the fraction of the problem’s unit tests that it passes. As item metadata we attach two reference solutions per problem, generated by the LLM-based method of Duan et al. (2026b).

We split the 50 items into five cross-validation folds as follows. We sort the items by their ground-truth difficulty and partition them into ten difficulty buckets of at most five items each; we then build a difficulty-stratified ordering by repeatedly drawing one item from each bucket in turn; rotating this ordering yields the five test sets (20% each), which partition all items. Within each fold, the remaining items are split 60/20 into train and validation, with validation taken at evenly spaced points along the difficulty-sorted remainder. This guarantees that every item is the test item of exactly one fold and that difficulty is balanced across the splits.

D Qualitative Examples

This appendix complements the θ -Align metric (Section 5.2) with a concrete, human-readable comparison. We show simulated student code at low, medium, and high conditioned ability. We annotate each snippet so the reader can directly compare how code quality tracks ability across sources. We use one cherry-picked CodeWorkout item (zeroMax, a fold-3 test item, 1B models). Human responses are real student submissions with ground-truth autograder scores; CAP and SFT responses are generated by the respective simulator and scored by our LLM scorer (the same one used for difficulty estimation). For generated code θ is the ability the simulator was conditioned on; for human code it is the student’s IRT-estimated ability (train-only fit).

Item. *Return a version of the given array where each zero is replaced by the largest **odd** value to its **right**; if there is no odd value to the right, leave the zero. A helper function may be used.*

Table 3 summarizes the scores: human and CAP rise monotonically with ability, whereas SFT does not.

Source	low θ	medium θ	high θ
Human (ground truth)	0.00	0.33	1.00
CAP (ours)	0.00	0.50	0.98
SFT	0.94	0.00	0.05

Table 3: Scores for the responses in this example (ground-truth for human, LLM-scorer-predicted for generated). Per-item Spearman(θ , score) is +0.57 (human), +0.64 (CAP), +0.17 (SFT).

Human (real students; ground-truth scores).

Low ($\theta=-1.26$, score 0.00): the helper correctly finds the largest odd value to the right, but the main loop never returns nums, so the attempt produces no result.

```
public int[] zeroMax(int[] nums) {
    for(int i = 0; i < nums.length; i++) {
        if(nums[i] == 0) { nums[i] =
            findBigOne(nums, i); }
    }
} // no return statement
private static int findBigOne(int[] numar,
    int position) {
    int biggest = numar[position];
    for(int i = position; i < numar.length; i
        ++){
        if(numar[i] % 2 != 0 && numar[i] >
            biggest) biggest = numar[i];
    }
    return biggest;
}
```

Medium ($\theta=-0.30$, score 0.33): finds the max odd to the right but never writes it back into nums[i], so only the “no odd to the right” cases pass.

```
public int[] zeroMax(int[] nums) {
    int oddValue = 0;
    for (int i = 0; i < nums.length; i++)
        if (nums[i] == 0)
            for (int k = i; k < nums.length;
                k++)
                if (nums[k] % 2 != 0 &&
                    oddValue < nums[k]) oddValue = nums[k];
    return nums; // oddValue is never
    written into nums[i]
}
```

High ($\theta=+2.17$, score 1.00): complete and correct, finding the max odd to the right, assigning it, and resetting the accumulator.

```
public int[] zeroMax(int[] nums) {
    int max = 0;
    for (int j = 0; j < nums.length - 1; j++)
        if (nums[j] == 0) {
            for (int i = j + 1; i <= nums.
                length - 1; i++)
                if (nums[i] > max && nums[i]
                    % 2 == 1) max = nums[i];
            nums[j] = max; max = 0;
        }
    return nums;
}
```

CAP (ours; predicted scores) mirrors the human trend. **Low** ($\theta=-1.76$, score 0.00): references maxOdd, which is never declared, and returns a fresh all-zero array, an incomplete attempt typical of a struggling student.

```
public int[] zeroMax(int[] nums)
{
    int[] a = new int[nums.length];
    for (int i = 0; i < nums.length; i++)
    {
        if (nums[i] == 0)
        {
            a[i] = maxOdd; // undefined
            variable
        }
    }
    return a;
}
```

Medium ($\theta=+0.11$, score 0.50): right shape (loop, cache, write) but the wrong rule, caching the last non-zero value to the *left* rather than the max odd value to the right.

```
public int[] zeroMax(int[] nums)
{
    int[] arr = new int[nums.length];
    int maxOdd = 0;
    for (int i = 0; i < nums.length; i++)
    {
        if (nums[i] == 0)
        {
            arr[i] = maxOdd; // uses
            the last non-zero seen to the LEFT
        }
        else
        {
            maxOdd = nums[i];
        }
    }
    return arr;
}
```

High ($\theta=+1.33$, score 0.98): cleanly factors out a findMaxOdd helper that scans $i+1$ rightward for odd values, the most ability-appropriate attempt of the three.

```
public int[] zeroMax(int[] nums)
{
    int[] result = new int[nums.length];
    for(int i = 0; i < nums.length; i++) {
        if(nums[i] == 0) {
            int max = findMaxOdd(nums, i + 1);

            if(max != 0) { result[i] = max; }
            else { result[i] = 0; }
        }
    }
    return result;
}
int findMaxOdd(int[] nums, int i)
{
    int max = -1;
    for(int j = i + 1; j < nums.length; j++)
        if(nums[j] % 2 != 0) max = Math.max(
            max, nums[j]);
    return max;
}
```

```
}
```

SFT (predicted scores) decouples quality from ability. Low ($\theta = -1.39$, score 0.94): a fully correct solution, despite the *low* conditioned ability.

```
public int[] zeroMax(int[] nums) {
    for (int i = 0; i < nums.length; i++)
        if (nums[i] == 0) {
            int maxOdd = 0;
            for (int j = i + 1; j < nums.
length; j++)
                if (nums[j] % 2 != 0) maxOdd
= Math.max(maxOdd, nums[j]);
            nums[i] = maxOdd;
        }
    return nums;
}
```

Medium ($\theta = -0.25$, score 0.00): undeclared maxOdd; returns all zeros.

```
public int[] zeroMax(int[] nums) {
    int[] array = new int[nums.length];
    for (int i = 0; i < nums.length; i++)
        if (nums[i] == 0) { array[i] = maxOdd;
    } // undefined variable
    return array;
}
```

High ($\theta = +0.85$, score 0.05): broken at *high* conditioned ability, taking the first non-zero (not odd) value and scanning from the start, not to the right.

```
public int[] zeroMax(int[] nums) {
    int[] result = new int[nums.length];
    for (int i = 0; i < nums.length; i++)
        if (nums[i] == 0)
            for (int j = 0; j < nums.length;
j++)
                if (nums[j] != 0) { result[i]
= nums[j]; break; }
    return result;
}
```

Reading the example. At low ability, both real students and CAP produce incomplete code with novice errors (a missing return; an undeclared variable); at medium ability the structure is right but a subtle bug remains (the result is never written back, or the scan runs the wrong direction), earning partial credit; at high ability the code is clean and correct, often factoring a helper. CAP thus reproduces the *ability* $\rightarrow$ *code quality* $\rightarrow$ *score* gradient of real students, the qualitative counterpart of its higher θ -Align (0.49 vs. 0.23 for SFT at 1B). SFT, in contrast, decouples quality from the conditioned ability: here its low-ability sample is fully correct while its high-ability sample is broken.

Caveats. This example is a single, cherry-picked item chosen for legibility, not aggregate evidence (the aggregate claim is carried by θ -Align); generated scores are *predicted* by the LLM scorer; the effect is shown at 1B, where CAP’s gains over

SMART occur; and the code is logically verbatim, with bugs intact, but reindented and annotated with explanatory comments for legibility.

E Prompts

We list the exact prompts below; $\{\cdot\}$ denotes a field filled in per item.

Simulated-student prompt (Response Gen SFT, SMART, and CAP).

System: You are a student of $\{\text{ability}\}$ ability responding to a stimulus.

User: Given the following ****Stimulus****, give a response to the ****question****:

****Stimulus****

Question:

$\{\text{question}\}$

Reference Solutions:

$\{\text{reference solutions}\}$

—

$\{\text{ability}\}$ is the IRT ability θ_j as a 4-decimal string; Response Gen ZS instead uses one of 10 descriptive buckets (“Minimal”–“Mastery”).

Automated-scoring prompt.

System: You are a teacher scoring student responses to open-ended questions.

User: Score the student response to the question by only outputting a single integer between 0 and 1 inclusive.

Question:

$\{\text{question}\}$

Reference Solutions:

$\{\text{reference solutions}\}$

—

Student response: $\{\text{response}\}$

Score:

The continuous score $\hat{y}_{ij} \in [0, 1]$ is read from the “0”/“1” token logits (Eq. 4), rather than by decoding the integer.

ICL difficulty-prediction prompt (ICL Random, ICL kNN).

System: You are an expert at estimating the difficulty of introductory programming questions. Difficulty is a real-valued IRT parameter centered near 0 (typical range about -2 to 2); higher means the question is harder for students. Reason briefly about how hard the question is, then end your answer with a line exactly of the form ‘Difficulty: <number>’.

User: Here are example questions with their known difficulties:

Example 1:

$\{\text{question}\}$

Reference Solutions:

$\{\text{reference solutions}\}$

Difficulty: $\{\text{difficulty}\}$

... (k in-context examples)

Now estimate the difficulty of this new question.
Reason step by step, then end with 'Difficulty:
<number>'.
{question}
Reference Solutions:
{reference solutions}

Each in-context item shows its question text and reference solutions; we parse the final “Difficulty:” value from the model’s output.

F AI Usage Acknowledgment

We used AI-based assistants to support brainstorming, paper writing, and code development.