SMART: Simulated Students Aligned with Item Response Theory for Question Difficulty Prediction

Alexander Scarlatos[1], Nigel Fernandez[1], Christopher Ormerod[2], Susan Lottridge[2], Andrew Lan[1]
[1]University of Massachusetts Amherst, [2]Cambium Assessment
{ajscarlatos,nigel,andrewlan}@cs.umass.edu
{christopher.ormerod,susan.lottridge}@cambiumassessment.com

Abstract

Item (question) difficulties play a crucial role in educational assessments, enabling accurate and efficient assessment of student abilities and personalization to maximize learning outcomes. Traditionally, estimating item difficulties can be costly, requiring real students to respond to items, followed by fitting an item response theory (IRT) model to get difficulty estimates. This approach cannot be applied to the cold-start setting for previously unseen items either. In this work, we present SMART (**Sim**ulated Students **A**ligned with I**RT**), a novel method for aligning simulated students with instructed ability, which can then be used in simulations to predict the difficulty of *open-ended* items. We achieve this alignment using direct preference optimization (DPO), where we form preference pairs based on how likely responses are under a ground-truth IRT model. We perform a simulation by generating thousands of responses, evaluating them with a large language model (LLM)-based scoring model, and fit the resulting data to an IRT model to obtain item difficulty estimates. Through extensive experiments on two real-world student response datasets, we show that SMART outperforms other item difficulty prediction methods by leveraging its improved ability alignment.

1 Introduction

Assessing student knowledge and skill levels, an important aspect of education, requires carefully designed assessment items (i.e., questions) with high quality and validity. According to the Standards for Educational and Psychological Testing (Eignor, 2013), various factors play a role in the quality and validity of items, with a crucial factor being their difficulty, i.e., how challenging they are to real students. Item difficulties also play a critical role in learning platforms, enabling personalized learning through student-tailored item recommendation (Chen and Duh, 2008; Ueno and Miyazawa, 2019), item generation (Jiao et al., 2023; Kurdi et al., 2020; Shimmei and Matsuda, 2022; Ashok Kumar et al., 2023), and curriculum design (Mehrens and Phillips, 1987; Zhang et al., 2024) to maximize learning outcomes.

Since new items are constantly developed and used in practice, estimating their difficulty before assigning them to students is crucial. Traditionally, estimating item difficulties can be costly and time-consuming (AlKhuzaey et al., 2024): we need (possibly a large number of) real students to respond to them, and then fit an item response theory (IRT) model on their responses to get item difficulty estimates. In contrast, recent work has estimated item difficulties from their (mostly textual) content (Tack et al., 2024; Gombert et al., 2024; Benedetto, 2023; Huang et al., 2017a), with some methods leveraging large language models (LLMs) (He-Yueya et al., 2024; Duenas et al., 2024). This approach works well for many items, especially multiple-choice questions (MCQs). Among them, an interesting approach is to use *simulated students* (He-Yueya et al., 2024; Feng et al., 2025; Maeda, 2025) to generate responses to new items at scale, to estimate their difficulties. Initial work in this direction uses simulated students to model student responses to MCQs, followed by aggregating their predicted responses and obtaining item difficulty estimates.

Among different item types, open-ended items require students to write short answers, and are common in many domains including math, programming, and particularly in reading comprehension assessment; see Table 3 for an example. Since these items require a student to construct an entire answer, they may reveal deeper insights into student knowledge (Brown and Burton, 1978; Feldman et al., 2018; Smith III et al., 1994; Anderson and Jeffries, 1985) than other item types, such as MCQs. However, estimating the difficulties of open-ended items can be more challenging, since

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 25083–25106
November 4-9, 2025 ©2025 Association for Computational Linguistics

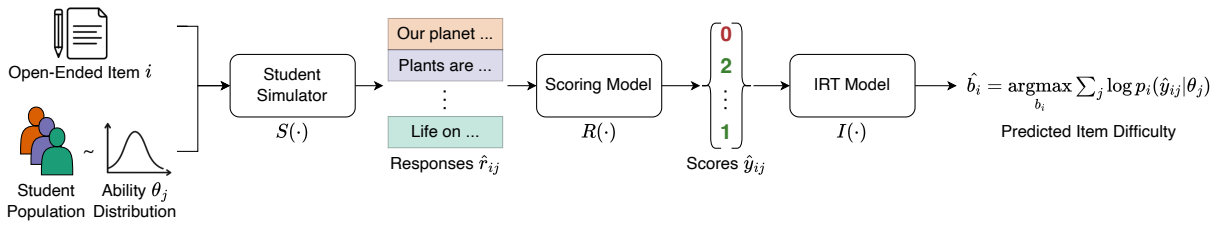

Figure 1: Our three-stage pipeline for item difficulty prediction: simulate open-ended responses for students with different abilities, score them, and fit an IRT model on the scores to obtain difficulty estimates.

scoring open-ended responses accurately at scale is not as easy as for MCQs. Moreover, there is considerable diversity in terms of both response content and style among real students, which means that it can be difficult to simulate student responses to these items (Sonkar et al., 2024).

1.1 Contributions

In this paper, we present SMART (**Sim**ulated Students **A**ligned with Item **R**esponse **T**heory), a pipeline for training LLMs as simulated students that generate open-ended responses, aligning them with an instructed ability level and item difficulty.

First, SMART leverages generative LLMs as the student simulator to synthesize *a distribution of* student responses to open-ended items. These responses provide valuable *interpretability* in the difficulty estimation process. We use direct preference optimization (DPO) to train our student simulator, using a novel method to form preference pairs. We *prefer student responses with higher likelihood under an IRT model*, thus aligning the generated responses with both student ability and item difficulty. Second, we use an LLM-based scoring model to score the generated responses. Finally, we train an IRT model on the scored student responses to obtain item difficulty estimates.

We conduct extensive quantitative experiments on two real-world datasets, one with 49 short-answer reading comprehension items and another with 50 Java coding items. We find that SMART, by simulating IRT-aligned responses to unseen items from a population of students with different abilities, outperforms state-of-the-art item difficulty prediction methods, even in the low-resource setting with few training items. We analyze the simulated student responses and find that SMART responses correlate strongly with their instructed ability, and that the feature distributions of its responses match closely with the ground-truth. We also qualitatively compare simulated student responses across methods, perform a failure pattern analysis, and suggest directions for future work.

2 Background: Item Response Theory

Item response theory (IRT) (Rasch, 1993) is a framework for jointly estimating scalar-valued abilities of students and parameters of "items" (or questions), mainly difficulty and discrimination, from student responses to items. In the simplest case, each student is represented by a single scalar ability $\theta_j \in \mathbb{R}$, each item is represented by a single scalar difficulty $b_i \in \mathbb{R}$, and each score y_{ij} is binary, i.e, 0 (incorrect) or 1 (correct). This simple model is known as the 1-parameter logistic (1PL) model (Rasch, 1993), and uses the following formulation:

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

In practice, θ_j and b_i are assumed to be drawn from a standard normal, with implementations using expectation maximization or variational inference (Lalor et al., 2019) to enforce standard normal priors on these variables.

In our setting, we study *short-answer* questions where the response is open-ended. We assume that students can receive partial credit on responses, leading to ordinal or continuous scores. For ordinal response data, we use the generalized partial credit model (GPCM) (Muraki, 1992), which adds a "step" parameter for each of the C total score categories:

$$P_i^{\text{GPCM}}(y_{ij}|\theta_j) = \frac{e^{\sum_{y=0}^{y_{ij}} a_i(\theta_j - b_i + d_{iy})}}{\sum_{c=0}^{C-1} e^{\sum_{y=0}^{c} a_i(\theta_j - b_i + d_{iy})}}, \tag{2}$$

where the response $y_{ij} \in \{0, \ldots, C-1\}$, d_{iy} is the step parameter for item i for score y, and a_i is the discrimination parameter for item i. GPCM smoothly adjusts the probability of different ordinal scores according to student ability and item difficulty, with step parameters determining where

the probability mass shifts from one score to the next.

We also handle continuous scores, i.e., $y_{ij} \in [0, 1]$, where the student's response lies on a spectrum between incorrect and correct. In this setting, we model correctness probability using a continuous Bernoulli distribution (Loaiza-Ganem and Cunningham, 2019):

$$P_i^{\text{Bern.}}(y_{ij}|\theta_j) = P_i^{\text{1PL}}(Y = 1|\theta_j)^{y_{ij}} \cdot P_i^{\text{1PL}}(Y = 0|\theta_j)^{(1-y_{ij})}. \quad (3)$$

3 Problem Formulation

We assume that we have access to a real-world dataset of human-scored student responses to open-ended short-answer items. Each item, indexed by i, is associated with 1) a short question text q_i, 2) (optional) meta data m_i, e.g., a passage text for reading comprehension items, 3) (optional) a scoring rubric w_i, and 4) a set of human-scored student responses denoted by $D_i = \{r_{ij}, y_{ij}\}$. An IRT model is fit to this data to get ground-truth estimates of each student ability θ_j and item difficulty b_i. Our goal is to predict the item difficulty parameters of previously *unseen* test items before students respond to them.

4 Methodology

Our approach, SMART, illustrated in Figure 1, uses a novel, three-stage pipeline, which simulates student responses to short-answer, open-ended items, to help estimate their difficulty. First, SMART leverages human-scored responses by real students to *open-ended items* to learn a student response simulator, $S(\cdot)$, parameterized by a generative LLM. The simulator $S(\cdot)$ generates a *distribution of student responses* to unseen open-ended items. Second, an LLM-based scoring model, $R(\cdot)$, scores these generated responses. Third, an IRT model, $I(\cdot)$, is trained on the scored student responses to estimate item difficulties.

4.1 LLM-based Student Response Simulation

We first detail our LLM finetuning method for student simulation, resulting in a model we denote as $S(\cdot)$. We combine the question text q_i, the meta data text m_i, and the student ability θ_j in our input prompt, and train the LLM using supervised finetuning (SFT) to maximize the likelihood of the ground-truth student response r_{ij} as output. We convert θ_j to a string after rounding to 4 decimal places, which we find works better than binning values into descriptive buckets; we leave an exploration of more advanced methods for numeric inputs to LLMs, such as weighting model parameters (Wang et al., 2024), for future work. See the exact prompt format in Appendix F.

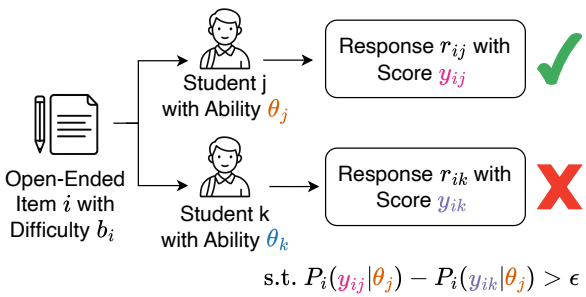

Figure 2: Our novel preference pair creation method through comparing response likelihood under IRT.

4.2 SMART: Ability Alignment in Response Simulation via Preference Optimization

We now motivate and detail SMART, our novel method aligning simulated student responses with their given ability and the expected difficulty of the given item. Even though the LLM-based student simulator $S(\cdot)$ is finetuned to maximize the likelihood of the textual content of student responses, we observe that the responses it generates do not always align well with the prompted ability value, likely because the SFT objective treats all output sequences equally without any special attention paid to the prompted ability.

To address this limitation, we propose the following idea: to incorporate information on the IRT likelihoods of student responses, in addition to their textual content. The intuition is that generating responses with high likelihoods under the IRT model should lead to better alignment between the IRT parameters and response content, which should benefit the downstream task of item difficulty prediction. To achieve this alignment, we use DPO (Rafailov et al., 2023) to *prefer responses with scores having a higher IRT likelihood* with the input student ability and item difficulty.

Preference Pair Creation. Our novel preference pair creation method is illustrated in Figure 2. Given an item's question text q_i, meta data m_i, and a student's ability θ_j, we compare two possible responses: one with a higher IRT likelihood, r_{ij}^w, which is preferred to another with a lower IRT

likelihood r_{ij}^{ℓ}, thereby forming a preference pair. We find such preference pairs among the human-scored real student responses. For each real student response r_{ij}, we construct a set of possible *negative candidates*, $\mathcal{R}_{ij}$, which are other responses to item i that a student with ability θ_j would be less likely to write, according to the IRT model. Formally:

$$\mathcal{R}_{ij} = \{r_{ik} : P_i(y_{ij}|\theta_j) - P_i(y_{ik}|\theta_j) > \epsilon\}, \quad (4)$$

where P_i is either P_i^{GPCM} (Eq. 2) if the scores are ordinal or $P_i^{\text{Bern.}}$ (Eq. 3) if the scores are continuous. The hyperparameter $\epsilon \in [0, 1)$ sets the threshold for how much *more* likely a response must be than another to form a preference pair, which is needed to reduce noisy preference pairs. In summary, for some student that responds with r_{ij}, for every other possible response, we ask the question: *how likely is it for the same student to write response* r_{ik}? We randomly select m negative candidates from $\mathcal{R}_{ij}$ for training, creating m preference pairs where the ground-truth response, $r_{ij}^{w} = r_{ij}$, is preferred over each sampled response, $r_{ij}^{\ell} = r_{ik}$. We note that $\mathcal{R}_{ij}$ could be empty in some cases, for instance, if the ground-truth student response is unlikely for the corresponding θ_j.

4.3 Scoring Model for Open-ended Responses

Given a distribution of varied generated student responses to items, we need to score them before fitting an IRT model to estimate item difficulties. This scoring task is challenging since it requires scoring synthetic responses to items not seen during training. We parameterize our scoring model $R(\cdot)$ by finetuning a generative LLM, given its ability to handle long-context inputs, to discern complex (partial) scoring criteria, optionally from a rubric, and to generalize to scoring responses to new items in the test set. We combine the question text q_i, the meta data m_i, the scoring rubric w_i, and the student response text r_{ij} into a prompt x_{ij}, and instruct the LLM to only output a single integer for the score label $y_{ij} \in \{0, \ldots, C-1\}$. For ordinal scores, we produce an estimate score $\hat{y}_{ij}$ via greedy decoding from $R(\cdot|x_{ij})$. For continuous scores, following (Scarlatos et al., 2025a), we smoothly interpolate between the token logits corresponding to fully correct and incorrect scores:

$$\hat{y}_{ij} = \frac{e^{R(1|x_{ij})}}{e^{R(0|x_{ij})} + e^{R(1|x_{ij})}}, \quad (5)$$

where $R(y|x_{ij})$ is the output logit value for token y returned by $R(\cdot|x_{ij})$, and we train using binary cross-entropy loss. See the exact prompt we use for our scoring model in Appendix F.

4.4 Simulation-Based Difficulty Prediction

We now detail how to use our ability-controlled student simulator and automated scoring model to predict item difficulties. First, we create a population of n simulated students such that their abilities match the training distribution: we create a 50-bucket histogram of abilities in the train set, proportionally draw n samples from the resulting distribution, and convert each sample to a random uniform value in the corresponding bin. Second, we perform the student simulation: for each drawn ability and item in the test set, we prompt the simulated student model $S(\cdot)$ to generate a corresponding response using nucleus sampling. Third, we score each generated response using the automated scoring model $R(\cdot)$. Finally, we re-fit the IRT model $I(\cdot)$ using the new set of scored responses. For each generated response, we assign a student ID by rounding the ability to the nearest decimal place, which we find to be crucial in preventing overfitting by reducing the number of ability parameters significantly. We provide additional details on IRT fitting in Appendix A.4. We take the difficulty estimates under this IRT model for the new, unseen items as our final predictions.

5 Experimental Evaluation

We now detail experiments that we conduct to validate our approach on real student response datasets.

5.1 Datasets

We conduct our experiments on two real-world open-ended student response datasets, each containing around 50 items and thousands of student responses.

Smarter Balanced. We first use a private real-world dataset, provided to us by Smarter Balanced[1]. The dataset contains student responses to open-ended short answer items administered to students in the United States, containing around 85K responses to 49 items from around 63K students. These open-ended items assess reading comprehension skills including argumentation, explanation, and narration, at the grade 6 level with the average length of student responses being 92 tokens. Each response is manually scored by a trained human scorer, using a rubric with detailed scoring criteria,

[1] https://smarterbalanced.org/

with scores in the set $\{0, 1, 2\}$. The meta data for each item includes a passage that the student must read before answering the question. Our trained 3B parameter scoring model achieves a quadratic weighted kappa (QWK) of 0.616 on this dataset, indicating moderate-to-high agreement with ground-truth scores, which is competitive with unseen item open-ended response scoring on other datasets[2]. We show example items in Tables 3 and 4.

CodeWorkout. We additionally use the publicly available CodeWorkout dataset (DataShop, 2021), originally used for the 2nd CSEDM Data Challenge (Organizers, 2021). The dataset, commonly used for student modeling in open-ended programming tasks (Liu et al., 2022; Duan et al., 2025), contains 50 Java coding problems in a first year college programming course, with student solutions across 2 semesters. Students can attempt a problem multiple times after receiving compiler feedback, so we only consider the first attempt for a problem for each student, resulting in 10,834 responses from 246 students. Each response is scored with a continuous value between 0 and 1, representing the portion of test cases that passed; the test cases themselves are not made available. To ensure that models understand how to correctly solve the problems, we include 2 LLM-generated solutions in each item's meta data, using the method described in (Duan et al., 2025). Our trained 3B parameter scoring model achieves a Pearson's correlation coefficient (PCC) of 0.849 on this dataset, indicating high agreement with ground-truth scores. We show an example item in Table 5.

We split each dataset into 5 cross-validation folds, with each fold split 60/20/20 item-wise for train/validation/test, respectively. We ensure that difficulties of items across all splits and folds are roughly even. We fit IRT models (P^{GPCM} for Smarter Balanced and $P^{\text{Bern.}}$ for CodeWorkout) on each fold to estimate ground-truth student ability and item parameters. For each fold, to ensure no information from unseen items is leaked at train time, we fit three separate IRT models, one on train data, one on train plus validation data, and one on all data. We use the estimated parameters from the three models to train our student simulator, and to perform validation and test-time evaluation, respectively. We provide further details on IRT fitting in Appendix A.4 and dataset processing in Appendix B.

[2]Best QWK on unseen items in NAEP Reading Automated Scoring Challenge was 0.528: https://github.com/NAEP-AS-Challenge/reading-prediction/blob/main/results.md

5.2 Metrics

Item Difficulty Prediction. Following prior work in item difficulty prediction (Huang et al., 2017a; He-Yueya et al., 2024), we use the widely adopted *Pearson Correlation Coefficient* (**PCC**) to measure the linear correlation between predicted and ground-truth item difficulties, *Spearman Rank Correlation Coefficient* (**SCC**) to measure the correlation between ranks of predicted and ground-truth item difficulties, and *Root Mean Squared Error* (**RMSE**) for a precision comparison between predicted and ground-truth item difficulties.

Student Simulation. We additionally propose metrics to investigate how faithful the generated responses are to real student ones. Since we model students at a population level, reference-based metrics such as ROUGE-L (Lin, 2004) and BERTScore (Zhang et al., 2020) are not suitable. Therefore, following prior work in population simulation (Bui et al., 2025; Yu et al., 2023), we report distribution-level metrics comparing the generated and ground-truth response distributions in a latent embedding space. We first extract embeddings for all responses using a sentence encoder (Reimers and Gurevych, 2019), specifically `multi-qa-mpnet-base-dot-v1`. We then compute the *Fréchet Inception Distance* (**FID**) (Heusel et al., 2017), which is common for evaluating generated images but has recently been adapted to text (Yue et al., 2023; Bui et al., 2025), and **MAUVE** (Pillutla et al., 2021), which measures the similarity of cluster distributions in the embedding space, to evaluate the alignment between the predicted and ground-truth student response distributions. Following (Bui et al., 2025), we also measure the similarity in diversity distribution by computing **Div. KL**, the KL divergence between the histograms of pairwise response cosine similarity across all ground-truth responses and across all simulated responses. Finally, we compute **θ-Align**, the Spearman correlation between input student ability and the predicted score of the corresponding generated response, measuring how well simulated students align with their prompted abilities.

5.3 Baselines

We compare our approach, SMART, to state-of-the-art item difficulty prediction approaches as well

as to several strong novel baselines adapted to our setting. These approaches fall into two categories: predicting difficulties 1) directly from item text, and 2) via student simulation.

Direct Difficulty Prediction Baselines. Following prior work (AlKhuzaey et al., 2024; Benedetto, 2023; Benedetto et al., 2023; Kapoor et al., 2025a; North and Ormerod, 2025), we adopt a strong finetuning baseline, **Diff SFT**, which finetunes a recent and capable encoder-based language model, specifically ModernBERT-base (Warner et al., 2024). We use a regression setup, minimizing mean squared error loss, to directly predict item difficulties given the item's question text and meta data. We also adopt a **Random Forest** baseline, which is common for difficulty prediction (AlKhuzaey et al., 2024; Gombert et al., 2024), where we encode the item's question text and meta data using the `multi-qa-mpnet-base-dot-v1` SBERT model (Reimers and Gurevych, 2019), and use the resulting embeddings as input features to fit a random forest regressor to item difficulty.

Since our set of items is small, we also introduce several baselines that do not require training. We use a simple yet strong baseline, **kNN Mean**, that retrieves the k items in the train set with the highest SBERT embedding cosine similarity, and takes their mean difficulty as the predicted test item difficulty. Similar methods have been used in automated scoring (Bin et al., 2008; Gomaa and Fahmy, 2012). We search over $k \in [1, 5]$ and select the best performing value on the validation set, selecting $k = 1$ for Smarter Balanced and $k = 5$ for CodeWorkout. Finally, we adopt two LLM prompting baselines leveraging in-context learning (ICL) (Brown et al., 2020), **ICL Random** and **ICL kNN**, where we prompt `Llama-3.1-70B-Instruct` with k in-context examples of items retrieved from the train set at random or using k-Nearest Neighbors (kNN) (Feng et al., 2024), respectively. We ask the model to first reason about the question's difficulty in a chain-of-thought manner before generating a numeric difficulty prediction. For kNN retrieval, we use the same SBERT model as above, and perform a similar hyperparameter search, selecting $k = 4$ for Smarter Balanced and $k = 2$ for CodeWorkout.

Student Simulation-based Difficulty Prediction Baselines. We compare SMART to other student simulation-based approaches for item difficulty prediction. For a fair comparison, we use the same base model and prompt template as SMART for the following methods, allowing these baselines to also serve as ablations. First, we employ **Response Gen ZS**, where we prompt an LLM in a zero-shot (ZS) manner to generate a student response given the item and ability. Since the model has not been trained on the distribution of numeric student abilities, we convert abilities to corresponding labels from 10 descriptive buckets shown in Appendix A.2, ranging from "minimal" to "mastery", with descriptions obtained by prompting Claude 3.5 Sonnet (Anthropic, 2024). This method is similar to the persona-based prompting method in (He-Yueya et al., 2024), where we use ability instead of student demographics. Second, we employ **Response Gen SFT**, as explained in Section 4.1, which finetunes the LLM on ground-truth student responses given the item and ability. Prior works that finetune simulated students generally include prior student responses in the prompt (He-Yueya et al., 2024; Zelikman et al., 2023); we use the student's ability as a proxy for this information.

5.4 Experimental Setup

We use `Llama-3.2-Instruct` (Grattafiori et al., 2024) from HuggingFace (Wolf et al., 2020) as the base model for our simulated student and automated scoring models. We perform all experiments with both the 1B and 3B parameter versions to examine how our methods scale across model sizes. We use the 1B automated scoring model to score responses from the 1B simulated student model, and do the same for 3B. We load base models with NF4 quantization (Dettmers et al., 2023) and finetune using LoRA (Hu et al., 2022). We provide further experimental details, including all hyperparameters, in Appendix A.1.

6 Results, Analysis, and Discussion

We now detail our quantitative results on item difficulty prediction and student response simulation, and qualitatively analyze the simulated student responses and SMART's strengths and weaknesses.

6.1 Quantitative Results

SMART outperforms other methods on item difficulty prediction. We report the average performance across all test items and cross-validation folds for both datasets in Table 1. Our approach, SMART, outperforms all direct prediction methods on all metrics, even largely maintaining this advantage when using the small 1B base LLM. It also

Model	Smarter Balanced			CodeWorkout		
	PCC ↑	SCC ↑	RMSE ↓	PCC ↑	SCC ↑	RMSE ↓
Direct Item Difficulty Prediction Methods						
Diff SFT (ModernBERT)	0.4937	0.3188	0.7627	−0.0425	−0.0036	0.7714
Random Forest (SBERT)	0.4021	0.3115	0.8623	0.1846	0.1418	0.6783
kNN Mean (SBERT)	0.5959	0.5219	0.6806	0.1949	0.2415	0.6701
ICL Random (Llama-3.1-70B)	0.1215	0.0348	1.0426	0.2487	0.1806	0.6462
ICL kNN (Llama-3.1-70B)	0.5273	0.4717	0.7634	0.0209	0.0821	0.7397
Student Simulation-based Item Difficulty Prediction Methods (Llama-3.2-1B)						
Response Gen ZS	0.2651	0.2436	0.9088	0.0733	0.0812	0.7226
Response Gen SFT	0.5621	0.4254	0.7174	0.2475	0.2388	0.6487
SMART (ours)	<u>0.6490</u>	0.4861	<u>0.6699</u>	0.3082	0.3188	0.6225
Student Simulation-based Item Difficulty Prediction Methods (Llama-3.2-3B)						
Response Gen ZS	0.3446	0.3963	0.8568	0.1487	0.1636	0.6912
Response Gen SFT	0.6069	**0.5830**	0.6905	**0.4256**	<u>0.3648</u>	**0.5627**
SMART (ours)	**0.6737**	<u>0.5661</u>	**0.6197**	<u>0.3929</u>	**0.4230**	<u>0.5890</u>

Table 1: Performance on question difficulty prediction. Best performance is in **bold** and second best is <u>underlined</u>. SMART outperforms all direct prediction methods and generally outperforms other simulation-based methods.

outperforms all other student simulation methods, with the exception of SFT when using 3B models; in these cases, SMART outperforms SFT by wide margins on some metrics while it is closely behind on others. These results indicate that both methods are powerful predictors of item difficulty, although there is high variability due to the small number of items in each dataset; larger test sets may be required to more clearly show SMART's advantage. We also observe that CodeWorkout is a more challenging dataset, with most methods performing worse than they do on Smarter Balance in terms of PCC and SCC. Still, the fact that SMART consistently outperforms baselines on both datasets shows that it can generalize across educational domains and student populations.

Student simulation is key when working with few training items. We observe that direct prediction baselines significantly underperform student simulation methods on both datasets. Diff SFT, which follows the approach of existing state-of-the-art methods (AlKhuzaey et al., 2024; Benedetto, 2023; Benedetto et al., 2023), performs moderately well on Smarter Balanced but poorly on CodeWorkout. This result is perhaps not surprising, since Diff SFT's training signal completely comes from given item difficulties, and thus can only work when a large number of training items are available. Direct difficulty prediction baselines that require no training also follow a similar pattern: kNN Mean and ICL kNN perform moderately on Smarter Balanced, likely because reading comprehension questions with similar semantic features are likely to pose similar tasks to the student and thus have similar difficulties. However, since this heuristic does not apply to coding questions, these baselines do not generalize across datasets. The poor performance of ICL Random also shows that powerful LLMs do not perform well on the difficulty prediction task without careful calibration. In our low resource setting with few items to train with, student simulation is key: student ability and item difficulty are only related via the response score when training an IRT model, but our student simulator can effectively couple IRT training with textual student responses and a scoring model. As a result, this extra information helps us to calibrate the IRT model better, making it easier to generalize to previously unseen items.

Ablation study. We compare the performance of Response Gen ZS, Response Gen SFT, and SMART, with all models parameterized with the same base LLM and prompts, serving as an ablation study. Response Gen SFT significantly outperforms Response Gen ZS, which suggests that finetuning is required for the LLM to adapt to the response distribution of real student responses. With the exception of some metrics when using 3B models, SMART outperforms Response Gen SFT, which validates our hypothesis that preferring responses with high IRT likelihoods in DPO training helps in the difficulty prediction task by calibrating the model in the context of student ability and implicit item difficulty.

SMART simulated student responses are aligned with abilities and diverse. Table 2 compares the student simulation methods on metrics

Model	Smarter Balanced				CodeWorkout			
	MAUVE ↑	FID ↓	Div. KL ↓	θ-Align ↑	MAUVE ↑	FID ↓	Div. KL ↓	θ-Align ↑
Student Simulation-based Item Difficulty Prediction Methods (Llama-3.2-1B)								
Response Gen ZS	0.0058	0.2713	0.2535	0.0399	0.0196	0.0747	0.0714	−0.0014
Response Gen SFT	0.1483	0.0847	0.0215	0.4481	0.1777	0.0269	0.0315	0.1730
SMART (ours)	0.1315	0.0918	0.0482	0.7338	**0.1931**	**0.0267**	**0.0284**	**0.5895**
Student Simulation-based Item Difficulty Prediction Methods (Llama-3.2-3B)								
Response Gen ZS	0.0052	0.3386	0.4032	0.0211	0.0042	0.1284	0.2779	−0.0101
Response Gen SFT	**0.1858**	**0.0757**	**0.0137**	0.5158	0.1426	0.0333	0.0515	0.2226
SMART (ours)	0.1713	0.0785	0.0279	**0.7655**	0.1479	0.0303	0.0335	0.5582

Table 2: Performance on student simulation metrics. Best performance is in **bold** and second best is underlined. SMART significantly outperforms other methods on ability following.

regarding the quality and faithfulness of simulated responses. We see that SMART outperforms other methods by a wide margin on the θ-Align metric, which measures the correlation between the desired input ability and inferred ability by scoring the generated responses. This observation shows that DPO training leads to a strong preference for the model to follow the input ability, resulting in a faithful student simulator. We also compare the simulated student methods on distribution similarity metrics, measuring how closely their generated responses align with ground-truth ones. Response Gen ZS performs poorly, which emphasizes the need for finetuning or DPO training, over simple prompting, to align the LLM with diverse responses by real students. Interestingly, Response Gen SFT is marginally better than SMART on these metrics on Smarter Balance, while SMART is better on CodeWorkout. This observation highlights the fact that DPO training prioritizes the IRT likelihoods, which may or may not align with how realistic the generated responses appear. We also observe that 1B models outperform 3B models on these metrics on CodeWorkout, perhaps due to the lower coding ability of the 1B model, therefore aligning better with code written by novice coders (students).

6.2 Qualitative Evaluation

To further understand the strengths and weaknesses of our method, we perform a qualitative analysis, where we examine patterns in items and generated responses that affect difficulty prediction accuracy.

Models Reflect Ability-Specific Features in Student Responses. Tables 3 and 4 show examples of generated responses to Smarter Balanced items for students with varying ability levels. Compared to SFT, SMART more clearly reflects realistic patterns based on ability. Broadly speaking, students with low ability have short one-sentence responses, spelling mistakes, poor grammar, and incorrect punctuation. High-ability students, on the other hand, have longer responses with complex sentence structures and strong vocabulary, and make an effort to answer all aspects of the question to receive full credit. SMART also captures student response diversity, especially for medium-ability students, whose responses usually contain simple words, but differ in length, coherence, and score obtained by the trained scoring model. Our preference pair creation encourages this diversity since varied responses can be preferred as long as they are sufficiently likely under the IRT model.

Table 5 shows examples of real and generated responses to a CodeWorkout item. SMART's difficulty prediction is much more accurate than SFT's on this item, primarily due to the diversity of generated responses; the SFT responses are semantically similar and receive similar scores across ability levels, while for SMART, low ability students score poorly while medium and high ability students score highly. Additionally, SMART more faithfully reflects semantic patterns in the student code, such as medium ability responses being longer and high ability responses being more concise and using more clear coding constructs.

Performance is Item-dependent. For Smarter Balanced items, we find that SMART's performance is highly dependent on the type of reading comprehension task (argumentation, narration, explanation, among others) in the item. In particular, difficulty estimates are accurate for items that require information recall or those that add narrative elements to a story. On the other hand, estimates are less accurate for items requiring higher order thinking (Lewis and Smith, 1993), such as summarizing the main idea of a passage, or writing an introduction to a paragraph that includes a "thesis"

or a "main controlling idea". For these items, we find that SMART tends to *overestimate* their difficulty. We see that even for high input ability values, SMART tends to generate incomplete responses that only state facts from the passage, rather than summarizing it. Such responses receive only partial credit and thus lead to high difficulty values under IRT. Future work can attempt to augment the training data with synthetic responses with higher-order thinking to aid simulation for high-ability students.

Outlier Generated Responses. We observe that SMART occasionally generates outlier responses with fidelity issues, with failure patterns including 1) repeating the passage, 2) not being relevant to the item, and 3) being highly repetitive. These outlier responses are often inaccurately scored by the scoring model, affecting downstream item difficulty prediction. A possible solution is to use a fidelity checking model to filter out such outlier responses and re-generate when it happens.

7 Related Work

Direct Item Difficulty Prediction Item difficulty prediction has been widely studied (AlKhuzaey et al., 2024), particularly in the context of multiple-choice questions (MCQs) (Yaneva et al., 2024; Feng et al., 2025; Reyes et al., 2023). It is common to train regression models using handcrafted features (Byrd and Srivastava, 2022; Tack et al., 2024; Kapoor et al., 2025b; Gombert et al., 2024), although finetuning language models such as BERT has proven to be highly effective (Gombert et al., 2024; Zhou and Tao, 2020; Loginova et al., 2021; Benedetto et al., 2021). Custom architectures, such as TACNN, have also been proposed for this task (Huang et al., 2017b).

Simulation-Based Difficulty Prediction An increasing body of research uses simulated students to estimate item difficulty. They follow a similar pipeline to our work, where they overgenerate a large set of responses using AI models, and then fit an IRT model to these responses to recover the item difficulty. Such methods have prompted ensembles of pre-trained language models to answer the items (Liu et al., 2025; Uto et al., 2024), finetuned language models to predict student responses given prior responses (Zelikman et al., 2023), and trained LSTMs with varying proportions of incorrect labels (Lalor et al., 2019). He-Yueya et al. (2024) experiment with LLM ensembles, finetuning, and persona-based prompting, finding that the most effective method depends on the dataset.

Our approach builds on this line of work but introduces several key innovations. First, we explicitly train simulated students to follow a prompted ability level, allowing the simulated students to align with the underlying IRT distribution. Second, we focus on open-ended responses, which are not handled by most finetuned or feature-based methods. Third, our system is designed to operate effectively on small item sets, where feature-based and BERT-based methods typically underperform.

8 Conclusions and Future Work

In this paper, we presented SMART, a novel method for aligning LLM-based simulated students with a prompted ability and implicit item difficulty via DPO training. The key is to form preference pairs where preferred responses are more likely under an IRT model. We perform a large-scale simulation with the resulting student model, automatically scoring generated responses with a trained LLM, and fit the resulting data with IRT to estimate the difficulties of unseen items. Through extensive experiments on a real-world student response dataset, we show that SMART outperforms other item difficulty prediction methods by capturing response patterns in the data that reflect the diverse abilities of the students and the difficulties of the items.

There are many avenues for future work. First, we can include generated student responses in the pool of candidates for DPO (Parikh et al., 2025), which may help to explicitly disprefer outlier responses that are not aligned with real student data. Second, we can explore the applicability of SMART in other domains such as programming, math, and in non question-response settings such as dialogues (Scarlatos et al., 2025b). Finally, we can extend SMART's student representation to go beyond simple scalar-valued student abilities and into more descriptive personas (Wu et al., 2025).

Acknowledgments

We thank Smarter Balanced for providing the private dataset used in our experiments. We also thank Joseph Di Garbo, Frank Rijmen, Suhwa Han, Benjamin Godek, Bokhee Yoon, Zhangqi Duan, and Jacqueline Scarlatos for helpful discussions around this work. This work is partially funded by the NSF under grants 2237676 and 2418657.

Limitations

There are several practical limitations to our work. First, while we experiment on two datasets across educational domains and student populations, there are limitations to these datasets. The Smarter Balanced dataset is not publicly available, and the CodeWorkout data takes place over the course of two semesters, making static ability modeling challenging. Additionally, the small number of items in each dataset makes reliable evaluation challenging. To the best of our knowledge, there are no other suitable publicly available open-ended testing datasets. Second, we restrict our student simulation to models no larger than 3B parameters, which is relatively small. While our methods are effective at this size, it would be helpful to see if scaling to larger sizes is worth the cost. Third, we find that the textual distribution of our generated responses does not always match the ground-truth responses, and that generated responses are occasionally degenerate. As a result, the pool of simulated responses may not look realistic from a human perspective. This limitation restricts our method's usefulness as a more generic student simulator, although it could be addressed by improvements to the method in future work.

Ethical Considerations

There are several potential societal benefits to our work. Primarily, accurate automated difficulty prediction can greatly benefit educational assessment: it can reduce the cost of pre-testing to calibrate item difficulties, it can reduce the risk of sensitive items being leaked from pre-tests, and it can enable calibrating difficulty for AI-generated personalized items, enabling more accessibility in assessments. There are several potential risks to our work as well. Incorrectly calibrated items could lead to unfair educational assessments if the calibrations are overfitted to a specific demographic. As bias is common in AI systems, simulated students may not sufficiently represent minority or non-English speaking students, thus leading to calibration errors for these populations. Future work should study these effects before deploying any such simulation-based difficulty prediction system.

References

Samah AlKhuzaey, 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.

John R Anderson and Robin Jeffries. 1985. Novice lisp errors: Undetected losses of information from working memory. *Human–Computer Interact.*, 1(2):107–131.

Anthropic. 2024. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet. Accessed: 2025-05-19.

Nischal Ashok Kumar, Nigel Fernandez, Zichao Wang, and Andrew Lan. 2023. Improving reading comprehension question generation with data augmentation and overgenerate-and-rank. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 247–259, Toronto, Canada. Association for Computational Linguistics.

Luca Benedetto. 2023. A quantitative study of nlp approaches to question difficulty estimation. In *International Conference on Artificial Intelligence in Education*, pages 428–434. Springer.

Luca Benedetto, Giovanni Aradelli, Paolo Cremonesi, Andrea Cappelli, Andrea Giussani, and Roberto Turrin. 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.

Luca Benedetto, Paolo Cremonesi, Andrew Caines, Paula Buttery, Andrea Cappelli, Andrea Giussani, and Roberto Turrin. 2023. A survey on recent approaches to question difficulty estimation from text. *ACM Computing Surveys*, 55(9):1–37.

Li Bin, Lu Jun, Yao Jian-Min, and Zhu Qiao-Ming. 2008. Automated essay scoring using the knn algorithm. In *2008 International Conference on Computer Science and Software Engineering*, volume 1, pages 735–738.

John Seely Brown and Richard R Burton. 1978. Diagnostic models for procedural bugs in basic mathematical skills. *Cogn. sci.*, 2(2):155–192.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Jason Brownlee. 2019. How to implement the frechet inception distance (fid) for evaluating gans. Accessed: 2025-05-19.

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.

Chih-Ming Chen and Ling-Jiun Duh. 2008. Personalized web-based tutoring system based on fuzzy item response theory. *Expert systems with applications*, 34(4):2298–2315.

DataShop. 2021. Dataset: Codeworkout data spring 2019. Online: https://pslcdatashop.web.cmu.edu/Files?datasetId=3458.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Zhangqi Duan, Nigel Fernandez, Arun Balajiee Lekshmi Narayanan, Mohammad Hassany, Rafaella Sampaio de Alencar, Peter Brusilovsky, Bita Akram, and Andrew Lan. 2025. Automated knowledge component generation and knowledge tracing for coding problems. *Preprint*, arXiv:2502.18632.

George Duenas, Sergio Jimenez, and Geral Mateus Ferro. 2024. UPN-ICC at BEA 2024 shared task: Leveraging LLMs for multiple-choice questions difficulty prediction. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 542–550, Mexico City, Mexico. Association for Computational Linguistics.

Daniel R Eignor. 2013. The standards for educational and psychological testing. *APA handbook of testing and assessment in psychology, Vol. 1. Test theory and testing and assessment in industrial and organizational psychology*.

Molly Q Feldman, Ji Yong Cho, Monica Ong, Sumit Gulwani, Zoran Popović, and Erik Andersen. 2018. Automatic diagnosis of students' misconceptions in K-8 mathematics. In *Proc. CHI Conf. Human Factors Comput. Syst.*, pages 1–12.

Wanyong Feng, Jaewook Lee, Hunter McNichols, Alexander Scarlatos, Digory Smith, Simon Woodhead, Nancy Ornelas, and Andrew Lan. 2024. Exploring automated distractor generation for math multiple-choice questions via large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3067–3082, Mexico City, Mexico. Association for Computational Linguistics.

Wanyong Feng, Peter Tran, Stephen Sireci, and Andrew Lan. 2025. Reasoning and sampling-augmented mcq difficulty prediction via llms. *Preprint*, arXiv:2503.08551.

Wael H Gomaa and Aly A Fahmy. 2012. Short answer grading using string similarity and corpus-based similarity. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 3(11).

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. 2017a. 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.

Zhenya Huang, Qi Liu, Enhong Chen, Hongke Zhao, Mingyong Gao, Si Wei, Yu Su, and Guoping Hu. 2017b. 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.

Ying Jiao, Kumar Shridhar, Peng Cui, Wangchunshu Zhou, and Mrinmaya Sachan. 2023. Automatic educational question generation with difficulty level controls. In *International Conference on Artificial Intelligence in Education*, pages 476–488. Springer.

Radhika Kapoor, Sang T. Truong, Nick Haber, Maria Araceli Ruiz-Primo, and Benjamin W. Domingue. 2025a. Prediction of item difficulty for

reading comprehension items by creation of annotated item repository. *Preprint*, arXiv:2502.20663.

Radhika Kapoor, Sang T Truong, Nick Haber, Maria Araceli Ruiz-Primo, and Benjamin W Domingue. 2025b. Prediction of item difficulty for reading comprehension items by creation of annotated item repository. *arXiv preprint arXiv:2502.20663*.

Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler, and Salam Al-Emari. 2020. A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education*, 30:121–204.

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.

Arthur Lewis and David Smith. 1993. Defining higher order thinking. *Theory into practice*, 32(3):131–137.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Naiming Liu, Zichao Wang, Richard Baraniuk, and Andrew Lan. 2022. Open-ended knowledge tracing for computer science education. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3849–3862.

Yunting Liu, Shreya Bhandari, and Zachary A. Pardos. 2025. Leveraging llm respondents for item evaluation: A psychometric analysis. *British Journal of Educational Technology*, 56(3):1028–1052.

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.

Ekaterina Loginova, Luca Benedetto, Dries Benoit, and Paolo Cremonesi. 2021. Towards the application of calibrated transformers to the unsupervised estimation of question difficulty from text. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 846–855, Held Online. INCOMA Ltd.

Hotaka Maeda. 2025. Field-testing multiple-choice questions with ai examinees: English grammar items. *Educational and Psychological Measurement*, 85(2):221–244.

William A Mehrens and Susan E Phillips. 1987. Sensitivity of item difficulties to curricular validity. *Journal of Educational Measurement*, 24(4):357–370.

Eiji Muraki. 1992. A generalized partial credit model: Application of an em algorithm. *Applied psychological measurement*, 16(2):159–176.

Eiji Muraki. 1993. Information functions of the generalized partial credit model. *ETS Research Report Series*, 1993(1):i–12.

Kai North and Christopher Ormerod. 2025. Cyborg data: Merging human with ai generated training data. *Preprint*, arXiv:2503.22736.

Challenge Organizers. 2021. The 2nd csedm data challenge. Online: `https://sites.google.com/ncsu.edu/csedm-dc-2021/`.

Nisarg Parikh, Alexander Scarlatos, Nigel Fernandez, Simon Woodhead, and Andrew Lan. 2025. LookAlike: Consistent distractor generation in math MCQs. In *Proceedings of the 20th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2025)*, pages 294–311, Vienna, Austria. Association for Computational Linguistics.

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. 1993. *Probabilistic models for some intelligence and attainment tests.* ERIC.

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.

Diego Reyes, Abelino Jimenez, Pablo Dartnell, Séverin Lions, and Sebastián Ríos. 2023. Multiple-choice questions difficulty prediction with neural networks. In *International Conference in Methodologies and intelligent Systems for Techhnology Enhanced Learning*, pages 11–22. Springer.

Alexander Scarlatos, Ryan S. Baker, and Andrew Lan. 2025a. Exploring knowledge tracing in tutor-student dialogues using llms. In *Proceedings of the 15th Learning Analytics and Knowledge Conference, LAK 2025, Dublin, Ireland, March 3-7, 2025*. ACM.

Alexander Scarlatos, Naiming Liu, Jaewook Lee, Richard Baraniuk, and Andrew Lan. 2025b. Training llm-based tutors to improve student learning outcomes in dialogues. In *Artificial Intelligence in Education*, pages 251–266, Cham. Springer Nature Switzerland.

Machi Shimmei and Noboru Matsuda. 2022. Automatic question generation for evidence-based online courseware engineering. In *iTextbooks@ AIED*, pages 18–25.

John P Smith III, Andrea A DiSessa, and Jeremy Roschelle. 1994. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *j. learn. sci.*, 3(2):115–163.

Shashank Sonkar, Naiming Liu, and Richard Baraniuk. 2024. Student data paradox and curious case of single student-tutor model: Regressive side effects of training LLMs for personalized learning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15543–15553, Miami, Florida, USA. Association for Computational Linguistics.

Anaïs Tack, Siem Buseyne, Changsheng Chen, Robbe D'hondt, Michiel De Vrindt, Alireza Gharahighehi, Sameh Metwaly, Felipe Kenji Nakano, and Ann-Sophie Noreillie. 2024. ITEC at BEA 2024 shared task: Predicting difficulty and response time of medical exam questions with statistical, machine learning, and language models. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 512–521, Mexico City, Mexico. Association for Computational Linguistics.

Maomi Ueno and Yoshimitsu Miyazawa. 2019. Uniform adaptive testing using maximum clique algorithm. In *Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part I*, page 482–493, Berlin, Heidelberg. Springer-Verlag.

Masaki Uto, Yuto Tomikawa, and Ayaka Suzuki. 2024. Question difficulty prediction based on virtual test-takers and item response theory. *Workshop on Automatic Evaluation of Learning and Assessment Content*.

Kaiwen Wang, Rahul Kidambi, Ryan Sullivan, Alekh Agarwal, Christoph Dann, Andrea Michi, Marco Gelmi, Yunxuan Li, Raghav Gupta, Kumar Avinava Dubey, Alexandre Rame, Johan Ferret, Geoffrey Cideron, Le Hou, Hongkun Yu, Amr Ahmed, Aranyak Mehta, Leonard Hussenot, Olivier Bachem, and Edouard Leurent. 2024. Conditional language policy: A general framework for steerable multi-objective finetuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2153–2186, Miami, Florida, USA. Association for Computational Linguistics.

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.

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. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Tao Wu, Jingyuan Chen, Wang Lin, Mengze Li, Yumeng Zhu, Ang Li, Kun Kuang, and Fei Wu. 2025. Embracing imperfection: Simulating students with diverse cognitive levels using LLM-based agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9887–9908, Vienna, Austria. Association for Computational Linguistics.

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.

Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. 2023. Large language model as attributed training data generator: a tale of diversity and bias. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Xiang Yue, Huseyin Inan, Xuechen Li, Girish Kumar, Julia McAnallen, Hoda Shajari, Huan Sun, David Levitan, and Robert Sim. 2023. Synthetic text generation with differential privacy: A simple and practical recipe. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1321–1342, Toronto, Canada. 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.

Haotian Zhang, Shuanghong Shen, Bihan Xu, Zhenya Huang, Jinze Wu, Jing Sha, and Shijin Wang. 2024. Item-difficulty-aware learning path recommendation: From a real walking perspective. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4167–4178.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Ya Zhou and Can Tao. 2020. Multi-task bert for problem difficulty prediction. In *2020 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pages 213–216.

A Additional Experimental Details

A.1 Student Simulation Hyperparameters

We perform a preliminary hyperparameter search on our validation sets. We set LoRA's rank = 128, α = 64, and dropout = 0.05. For Smarter Balanced, for both the scoring model and student model, we perform SFT for 1 epoch, with learning rate = 1e-4 and linear warmup for 10% of steps, weight decay = 1e-2, gradient clipping norm = 1.0, and effective batch size = 64 using gradient accumulation. For DPO, we set learning rate = 1e-5, β = 0.5, the probability threshold $\epsilon = 0.1$ (Eq. 4), the number of negative candidates $m = 3$, and train on a random 20% subset of the training set to reduce costs. For CodeWorkout, we use the same hyperparamters except for the following: for the scoring model we perform SFT for 5 epochs with learning rate = 5e-5, for the student model we perform SFT for 3 epochs with learning rate = 1e-5, and perform DPO for 1 epoch with learning rate = 1e-6 on the full training set. We conduct all experiments on the Smarter Balanced dataset on NVIDIA A10G GPUs and all experiments on the CodeWorkout dataset on NVIDIA A40 GPUs.

At inference time, we sample $n = 1,000$ student abilities for simulation and generate a single response for each item/ability pair. We decode up to 500 tokens with nucleus sampling, setting temperature = 0.7 and p = 0.95. We use an inference-time batch size of 64 for the simulated student model and 2 for the scoring model, optimizing for inference speed.

On the Smarter Balanced dataset, for both SFT and DPO, training the simulated student model takes approximately 3 hours on a single GPU for the 1B model and 6 hours on 2 GPUs for the 3B model. Training the automated scoring model takes approximately 4 hours on a single GPU for the 1B model and 9 hours on 2 GPUs for the 3B model. Generating 10,000 responses (1,000 abilities over 10 items) takes approximately 1 hour on a single GPU for the 1B model and 3 hours on a single GPU for the 3B model. Scoring these responses takes approximately 15 minutes on a single GPU for the 1B model and 35 minutes on a single GPU for the 3B model.

A.2 Additional Implementation Details

We use `multi-qa-mpnet-base-dot-v1` for our SBERT model since it is the highest performing model with a 512 token limit, according to the sentence transformers model page[3]. We also found that it worked better than `all-distilroberta-v1` in preliminary experiments.

We train our random forest baseline using the RandomForestRegressor class from the sklearn[4] library. We additionally restrict max_features to the size of train set to avoid overfitting, which we found improves performance.

For our Response Gen ZS student simulation baseline, we first normalize all train abilities to $[-3, 3]$ before sampling, and then map abilities to 10 labeled buckets with the following ranges: "Minimal": $\theta < -2.5$, "Emerging": $\theta \in [-2.5, -2)$, "Developing": $\theta \in [-2, -1.5)$, "Approaching Proficiency": $\theta \in [-1.5, -0.8)$, "Proficient": $\theta \in [-0.8, 0)$, "Competent": $\theta \in [0, 0.8)$, "Skilled": $\theta \in [0.8, 1.5)$, "Advanced": $\theta \in [1.5, 2.0)$, "Exceptional": $\theta \in [2, 2.5)$, "Mastery": $\theta \geq 2.5$.

For computing MAUVE, we use the official implementation[5]. For computing FID, we adapt the implementation from (Brownlee, 2019).

A.3 Predicted Difficulty Normalization

For all methods, including baselines, we transform predicted difficulties to have the same mean and standard deviation as the ground-truth difficulties on the train set; doing so improves RMSE when predicted difficulties are correlated but distributionally shifted from the ground-truth. We perform the transformation using the following formula:

$$\hat{b}_i \leftarrow \frac{\sigma_2}{\sigma_1}(\hat{b}_i - \mu_1) + \mu_2, \tag{6}$$

where $\hat{b}_i$ is the predicted difficulty for item i, μ_1 and σ_1 are the mean and standard deviation of the predicted item difficulties, respectively, and μ_2 and σ_2 are the mean and standard deviation of the item difficulties in the train set, respectively.

A.4 IRT Model Fitting

We perform IRT fitting at 2 distinct stages in our pipeline: 1) when fitting to real student responses to create the ground-truth ability and item parameters,

[3] https://sbert.net/docs/sentence_transformer/pretrained_models.html#original-models

[4] https://scikit-learn.org/

[5] https://github.com/krishnap25/mauve

and 2) when fitting to simulated student responses to produce difficulty estimates.

The Smarter Balanced dataset does not contain student identifiers, but rather contains prior ability estimates based on each student's responses to a larger set of questions, including other types such as multiple choice questions; the 49 short answer questions we use comprise all short answer items in the full set. To ensure that other item types and test set responses do not influence our ability estimates, we fit an IRT model from scratch only on the 49 items used in this study. Since we do not have access to student identifiers, we use the prior ability as a proxy. However, the student responses are very sparse on our set of questions, with around 63K unique ability values for around 85K responses, leading to a poor fit for the IRT model. As a solution, we round each prior ability to a single decimal place (we found this precision to work best out of several tried) and use the resulting value as the student identifier. This transformation leads to a significantly better fit for the ground-truth IRT model. We note that this process assumes access to prior estimates of student ability, which may or may not be available in real-world settings. However, we note that this a limitation of the data, rather than our method. Additionally, we note that this transformation does not unfairly benefit our method over the baselines, so the comparison between methods in this work remains fair.

When we fit the IRT model with the simulated responses, we perform a similar operation for assigning identifiers. We round the prompted ability for each simulated response to one decimal place, and use this value to form a new identifier for all simulated responses that result in that rounded ability. When re-training the IRT model, we include the ground-truth student responses from the train set only. While these ground-truth student responses do not directly interact with the estimated parameters for non-train items or simulated students, we find their inclusion to improve performance, perhaps due to their effect on global variables such as AdamW's adaptive learning rate.

We implement our IRT models using PyTorch, using the formulations in Equations 2 and 3 to compute the probability of a student's score for a particular item. For GPCM, following prior work (Muraki, 1993), we set the first step parameter to $d_{i0} = 0$, since only $C-1$ step parameters are required for C score classes. We also impose the location constraint (Muraki, 1993), where all step parameters for an item sum to 0, i.e., $\sum_{y=0}^{C-1} d_{iy} = 0$, ensuring that difficulty parameters across items are on the same scale and are identifiable. We also fix all discrimination parameters to 1, which improves model fit. We do not enforce normal priors on any parameters. We train the model using gradient descent, using the cross-entropy loss between the predicted score probabilities and the ground-truth scores. We train using the AdamW optimizer for 50 epochs with learning rate = 1e-3, weight decay = 0, and batch size = 256; for CodeWorkout, we train for 200 epochs when fitting the *ground-truth* data, which is necessary due to the smaller number of ground-truth student responses. We reserve a random validation set of 20% of all responses to determine how predictive our ground-truth IRT parameters are, comparing the most likely student scores predicted by the IRT model and the ground-truth scores in the validation set. For Smarter Balanced, we measure performance using QWK since scores are ordinal, with the IRT model achieving a QWK of 0.57 on the validation set. For CodeWorkout, we measure performance using PCC since scores are continuous, with the IRT model achieving a PCC of 0.49. These values show moderate predictability, indicating that while the student scores are difficult to predict, the parameters are well-calibrated.

B Additional Dataset Details

B.1 Cross-Validation Splitting

We create our cross-validation splits such that 1) each item appears in train, validation, and test at least once, and 2) the distribution of item difficulties should be roughly the same across train, validation, and test over all folds. We achieve these requirements using the following algorithm. First, we sort all items according to their difficulty. Second, we partition all items into 10 difficulty buckets of up to 5 items each. Third, we sequentially construct a list of items by adding one random item at a time from each bucket in a round-robin fashion; the resulting list is striped by difficulty bucket. Finally, we evenly rotate this list 5 times to form the 5 cross-validation folds, partitioning each rotation at 60% and 80% to create a train/validation/test split.

B.2 Passage Summarization

Several Smarter Balanced questions include long reading passages, up to $4,000$ tokens in length, leading to increased memory usage and training time; therefore, we prompt

`Llama-3.1-70B-Instruct`, loaded with NF4 quantization, to summarize these passages while retaining the main elements and original writing style. We show this prompt in Appendix F.

C Data Scaling Experiment

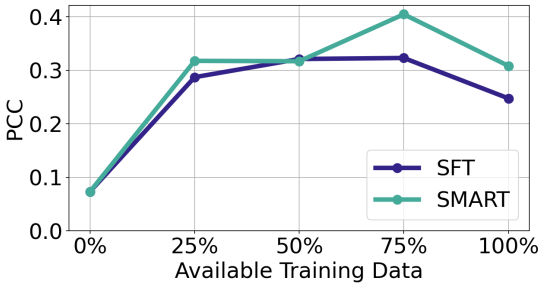

Figure 3: Results of data scaling experiment. Even a small amount of data is sufficient to achieve high performance. SMART begins to outperform SFT more significantly when more data is available.

We conduct an experiment where we vary the amount of data available for training by using random subsets of the student responses in the training set at 0%, 25%, 50%, 75%, and 100%. We retrain both SFT and SMART using the 1B model on these subsets, and report the PCC averaged over all cross-validation test splits on CodeWorkout. To only examine the impact of training data on the student model, we use the scoring model trained on all data, since in reality a stronger scoring model may be available (such as one that runs test cases on generated code).

We show the results of this data scaling experiment in Figure 3. We observe that even a small amount of training data is sufficient to drastically increase performance from zero-shot (25% of the training data contains 1,636 responses and 3,027 preference pairs in total). SFT and SMART perform similarly until 75%, when SMART begins to outperform SFT. Surprisingly, performance drops for both methods from 75% to 100%, indicating that some training responses may negatively impact performance, such as ones that represent item types that are not present in the test set. This result further reflects the challenges of testing on a small number of items.

D Statement on Scientific Artifacts

Software and Models To the best of our knowledge, all software and models we build our implementation on have open-source licenses or no available license. Additionally, we are within their intended terms of use, specifically, scientific research. If we release code, we will ensure the license and terms reflect the sources we build on.

Dataset The Smarter Balanced dataset contains proprietary questions and private student data. We do not examine the data for offensive content or identifiable information, because 1) the data is private and securely stored, 2) all models trained on the data are securely stored and will not be released, and 3) the data is very large and would be difficult to effectively scan for such issues. We do not check for such issues on CodeWorkout either, because it is a public and widely used dataset, and such investigations are beyond the scope of this work.

E Qualitative Examples

Passage	Source #1 - Meet the Robots Robots are not just a thing of the future; they are already doing many jobs today... Source #2 - Robots That Play Well with Others Robots are not just for work; they can also play. Some robots can sing, dance, and even play music... Source #3 - When the Car Is the Driver The state of Nevada has finalized new rules that will make it possible for robotic self-driving cars to receive their own special driving permits...
Question	Many robots are designed to do normal tasks that improve people's lives or jobs. Provide two pieces of evidence from different sources that support this idea and explain how each example supports the idea.
Ability	Response
	SFT
Low	The reason why these are good robots is because they can help with anything that you need. They can help with food, water, and other things that you need. They can also help with work, like doctors, and other things that you need to do.
Medium	In source 1 it talks about a robot named "Gemini Scout" that is a remote controlled robot. It can also be used for mine, farms, and doctors. It can also help people who are in danger. It can also help with rescues. In source 2 it talks about...
High	A few robots that are designed to do normal tasks that improve people's lives or jobs are, robots like the "Gemini-Scout" which can be used to rescue people that are in danger, such as in a mine or a building. It can also be used to test the air for dangerous gases. The "Gemini-Scout" is very useful to people because it helps them in many different ways. Another robot that is useful...
	SMART (ours)
Low	The reason why i think the robots are good is becuase they can help and they are smart.
Medium	In source 1 it talks about a robot named "Gemini Scout" that can carry food and water. It can also collect information about the air that comes out of the vents in the volcano. In source 2 it talks about...
High	A large number of robots are designed to help people with tasks that make their lives easier and to improve job efficiency. A piece of evidence from "Meet the Robots" is a robot that is used in mines to help search for lost or injured people. The robot is called the Gemini-Scout, and it is a remote-controlled robot. This supports the idea because it is helping to rescue people who are injured or missing...

Table 3: Examples of student responses generated by SFT and SMART for a publicly available Smarter Balanced item, conditioned on abilities in the low, medium, and high ranges. SMART adapts better to the instructed ability than SFT. For low ability, SMART introduces a spelling error and poor capitalization, and does not back up its claim, while SFT does not answer the question correctly but uses better sentence structure. For medium ability, both methods provide evidence but do not explain how they support the claim. For high ability, SMART produces structured writing, provides evidence, and explains how its evidence supports the claim, while SFT is less explicit about how its evidence supports the claim.

Passage	Plants need food to survive, just as animals do; but, unlike animals, they do not eat solid foods. Instead, plants get most of the food they need from the sun through a process called photosynthesis. Using this process, plants can absorb the sun's light energy to make their own food. Although they cannot move around to find food, plants can make their food anywhere as long as they have three things: carbon dioxide, water, and light...
Question	Write an introduction to the report that establishes and introduces a clear claim about how plants make their own food.
Ability	Response
	SFT
Low	Photosynthesis is the process by which plants make their own food from the sun.
Medium	Plants are very important to us because they give us oxygen, that we need to survive. Plants are very important because they give us oxygen.
High	Plants are very important to us. We rely on them for food and for oxygen. Plants make all of this possible. They take the carbon dioxide, the water, and the light, and turn them into sugar and oxygen. That is why photosynthesis is so important to animals.
	SMART (ours)
Low	that the plants are good for us and the world
Medium	Plants are one of the most important things on earth, without them we would not be alive today. Plants are very important to us humans because without them we wouldn't be able to survive, they also help us in many ways like we can make food out of them and plants help us by providing oxygen.
High	Photosynthesis is the process by which plants create food. The process of photosynthesis is very important to all living things, especially because without it, the air we breathe would be filled with only carbon dioxide, and without oxygen, animals and humans would not be able to survive. In this report, you will learn more about how plants make their own food.

Table 4: Examples of student responses generated by SFT and SMART for a publicly available Smarter Balanced item, conditioned on abilities in the low, medium, and high ranges. SMART adapts better to the instructed ability than SFT. For low ability, SMART uses very simple language and doesn't discuss how plants make food, while SFT uses more formal language and does mention how plans make food, which would be atypical for a low ability student. For medium ability, both methods write a coherent introduction but don't mention how plants make food; SMART writes a slightly longer response. For high ability, SMART makes sure to motivate the importance of photosynthesis as well as directly answer the question by stating how plants make food, while SFT neglects to mention how plants make food.

Question	Return the centered average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division to produce the final average. You may assume that the array has a length of 3 or more.	
Ability	Response	Score
	Human Student	
Low		0.0

```
public int centeredAverage(int[] nums)
{
    int merlin = 0;
    int less = nums[0];
    int more = nums[0];

    for (int i = 0; i < nums.length; i++)
    {
        merlin = merlin + nums[i];
        less = Math.min(less, nums[i]);
        more = Math.max(more, nums[i]);
    }
    return (sum - (min-max)/nums.length);
}
```

Medium 0.67

```
public int centeredAverage(int[] nums)
{
    int max = 0;
    int min = nums[0];

    for (int i = 0; i < nums.length; i++)
    {
        if (nums[i] > max)
        {
            max = nums[i];
        }

        else if (nums[i] < min)
        {
            min = nums[i];
        }
    }

    int sum = 0;
    int ct = 0;

    for (int n : nums)
    {
        if (n != min && n != max)
        {
            sum += n;
            ct++;
        }
    }
    return sum/ct;
}
```

High 1.0

```
public int centeredAverage(int[] nums)
{
    int sum = 0;
    int min = nums[0];
    int max = nums[0];

    for(int i = 0; i < nums.length; i++) {
        sum += nums[i];
        min = Math.min(min, nums[i]);
        max = Math.max(max, nums[i]);
    }

    return (sum - min - max) / (nums.length - 2);
}
```

SFT

Low 0.68

```
public int centeredAverage(int[] nums)
{
    int sum = 0;
    for (int i = 0; i < nums.length - 2; i++)
    {
        sum = sum + nums[i];
    }
    return sum / (nums.length - 2);
}
```

Medium 0.75

```
public int centeredAverage(int[] nums)
{
    int sum = 0;
    for (int i = 0; i < nums.length; i++)
    {
        sum = sum + nums[i];
    }
    int avg = sum / (nums.length - 2);
    return avg;
}
```

High 0.73

```
public int centeredAverage(int[] nums)
{
    int sum = 0;

    for (int i = 0; i < nums.length; i++) {
      if (nums[i]!= Integer.MIN_VALUE && nums[i]!= Integer.MAX_VALUE)
            sum += nums[i];
    }

    return sum / (nums.length - 2);
}
```

SMART (ours)

Low 0.27

```
public int centeredAverage(int[] nums)
{
    int i = 0;
    int sum = 0;
    for (int i = 0; i < nums.length; i++)
    {
        sum = sum + nums[i];
    }
    return sum / nums.length - 1;
}
```

Medium 0.94

```
public int centeredAverage(int[] nums)
{
    int sum = 0;
    int min = Integer.MAX_VALUE;
    int max = Integer.MIN_VALUE;

    for (int i = 0; i < nums.length; i++) {
        if (nums[i] > max) {
            max = nums[i];
        }
        if (nums[i] < min) {
            min = nums[i];
        }
        sum += nums[i];
    }

    return (sum - max - min) / (nums.length - 2);
}
```

High 0.98

```
public int centeredAverage(int[] nums) {
  int min = Integer.MAX_VALUE;
  int max = Integer.MIN_VALUE;
  int sum = 0;

  for(int num : nums) {
    if(num < min) min = num;
    if(num > max) max = num;
    sum += num;
  }

  sum -= min + max;
  return sum / (nums.length - 2);
}
```

Table 5: Examples of real student responses and responses generated by SFT and SMART for a CodeWorkout item, conditioned on abilities in the low, medium, and high ranges. Scores for real students represent the portion of test cases that passed; scores for simulated students are assigned by our trained scoring model. The true difficulty for the question is -0.57, the SFT difficulty prediction is 0.22, and the SMART difficulty prediction is -0.41. SMART responses are much more diverse than SFT responses, likely resulting in the more accurate prediction. To further demonstrate this, we report the average score for each ability quartile from each source, where real students and SMART show a gradual increase in score with ability, while SFT does not show any distinction between abilities. Real students: [0.22, 0.53, 0.71, 0.95], SFT: [0.81, 0.79, 0.82, 0.83], SMART: [0.57, 0.79, 0.85, 0.93].

F Prompts

We show our prompts below for the simulated student model (Table 6), the automated scoring model (Table 7), ICL for difficulty prediction (Table 8), and summarizing reading passages (Table 9). We note that because the simulated student and automated scoring models are finetuned, their system prompts are much less detailed. Doing so reduces GPU memory, and is permissible since the models are able to learn to perform their tasks from the data, rather than relying on instructions in the prompt.

System	You are a student of {ability_level} ability responding to a stimulus.
User	Given the following **Stimulus**, give a response to the **question**: **Stimulus** *— for Smarter Balanced —* Passage: {passage} —— Question: {question} *— for CodeWorkout —* Reference Solutions: {reference solutions} ——

Table 6: Prompt for the simulated student model.

System	You are a teacher scoring student responses to open-ended questions.
User	Score the student response using the rubric to the question by only outputting a single integer between 0 and {max_score} inclusive. *— for Smarter Balanced —* Rubric: {rubric} Passage: {passage} —— Question: {question} *— for CodeWorkout —* Reference Solutions: {reference solutions} —— Student response: {response} Score:

Table 7: Prompt for the automated scoring model.

System	You are an experienced English teacher. Your job is to predict the difficulty of a reading comprehension short answer question. Please follow these instructions carefully: - You will be given a question and optionally an associated passage. Use both of these to predict the question's difficulty. - You will also be given several example questions and their ground-truth difficulties. Use these to calibrate your prediction for the current question. - Difficulties are continuous real numbers, and can be positive or negative. A higher value means the question is more difficult. - First briefly think step-by-step about the question's difficulty based on its content and the examples. - Write your output with the following template: "Thinking: <step-by-step reasoning>\nDifficulty: <real-valued difficulty prediction>". Ensure that you only output a number for the difficulty prediction.
User	#### Example 1 #### *— for Smarter Balanced —* Passage: {passage} ——— Question: {question} *— for CodeWorkout —* Reference Solutions: {reference solutions} ——— Difficulty: {difficulty} #### Example 2 #### ... #### Example 3 #### ... #### Current Question #### *— for Smarter Balanced —* Passage: {passage} ——— Question: {question} *— for CodeWorkout —* Reference Solutions: {reference solutions} ———

Table 8: Prompt for in-context learning difficulty prediction.

System	You are an English teacher. You will be given a reading passage. Your job is to rewrite the passage to make it shorter. Please follow these instructions carefully: - Your version of the passage should be significantly shorter. - It should contain all of the main ideas and plot elements of the original passage. - Do not change the writing style of the passage. - If the passage contains multiple texts/sources, rewrite all of them in independent paragraphs. - Do not write an answer to the question. - Your output will be used as the new passage. Therefore, only output the rewritten passage. Do not write any explanation of your output at all.
User	{passage}

Table 9: Prompt for passage summarization.